Diplomarbeit

Eine iterative Heuristik für aperiodische Fahrplangestaltung mit OD-Paaren

vorgelegt von

Jennifer Anhalt

aus

Duderstadt

angefertigt
im Institut für Numerische und Angewandte Mathematik
der Georg-Augst-Universität zu Göttingen

Januar 2012

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel “Eine iterative Heuristik für aperiodische Fahrplangestaltung mit OD-Paaren” selbständig, ohne fremde Hilfe und ohne Benutzung anderer als den angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Göttingen, Januar 2012, Jennifer Anhalt

Symbolverzeichnis

$\mathcal{A}_{change}$	die Menge der Umsteigeaktivitäten
$\mathcal{A}_{dest}$	die Menge der Destination-Kanten
$\mathcal{A}_{drive}$	die Menge der Fahraktivitäten
$\mathcal{A}_{org}$	die Menge der Origin-Kanten
$\mathcal{A}_{wait}$	die Menge der Warteaktivitäten
$\mathcal{A}$	die Menge der Fahr-, Warte- und Umsteigeaktivitäten
$\mathcal{A}'$	die Menge der Origin- und Destination-Kanten
$\mathcal{A}^+(u)$	die Menge der von Origin-Knoten u ausgehenden Origin-Kanten
$\mathcal{A}^-(v)$	die Menge der in Destination-Knoten v eingehenden Destination-Kanten
$\mathcal{A}_{con}$	die Menge aller Kanten, die zwei Zusammenhangskomponenten aus $\mathcal{N}'$ verbinden
$\mathcal{A}_{headway}$	die Menge der Headwayaktivitäten
$\mathcal{A}^k_{\text{virt}}$	die Menge der virtuellen Kanten für OD-Paar $(u_k, v_k) \in OD$
$\tilde{\mathcal{A}}$	die Menge aller Aktivitäten
A	die Menge der gerichteten Kanten in einem Graphen
$\mathcal{E}_{arr}$	die Menge der Ankunftsereignisse
$\mathcal{E}_{dep}$	die Menge der Abfahrtssereignisse
$\mathcal{E}_{dest}$	die Menge der Destination-Knoten
$\mathcal{E}_{org}$	die Menge der Origin-Knoten
$\mathcal{E}$	die Menge der Ereignisse
$\mathcal{E}'$	die Menge der Origin- und Destination-Knoten
$\mathcal{E}(u)$	die Menge der Abfahrtsereignisse für Origin-Knoten u
$\mathcal{E}(v)$	die Menge der Ankunftsereignisse für Destination-Knoten v
E	die Menge der ungerichteten Kanten in einem Graphen
err_{abs}	der absolute Fehler der Heuristik
err_{rel}	der relative Fehler der Heuristik
C^+	Menge der Vorwärtskanten in einem Kreis C
C^-	Menge der Rückwärtskanten in einem Kreis C
$\delta^+(v)$	die Menge der von $v \in V$ ausgehenden Kanten
$\delta^-(v)$	die Menge der in $v \in V$ eingehenden Kanten
$\Delta_a = [L_a, U_a]$	Span von a
$d^+(v)$	der Außengrad von $v \in V$
$d^-(v))$	der Innengrad von $v \in V$
$diff_a := U_a - L_a$	Differenz von oberer Schranke U_a und unterer Schranke L_a für Kante $a \in \mathcal{A}$

G	ein Graph
L_a	untere Schranke für Kante a
$LB_i, i = 0, \ldots, 3$	untere Schranken für das aperiodische Fahrplanproblem mit OD-Paaren
$\mathcal{N}$	Ereignis-Aktivitätsnetzwerk
$\mathcal{N}'$	Ereignis-Aktivitätsnetzwerk mit Origin/Destination-Knoten und -Kanten
$\mathcal{N}^*$	Netzwerk mit Vorwärts- und Rückwärtskanten, konstruiert aus $\mathcal{N}$
$\widetilde{\mathcal{N}'}$	Ereignis-Aktivitätsnetzwerk mit Origin/Destination-Knoten und -Kanten und Kanten $\mathcal{A}_{con}$
OD	Menge der OD-Paare
Π	Fahrplan
$RZ^{\Pi^k}_{w_a^j}$	Reisezeit bezüglich Fahrplan Π^k in Iteration k und Passagierverteilung w_a^j in Iteration j
$RZ^{\Pi^0}_{w_a^1}$	Reisezeit, die sich aus den Längen der Anfangslösung x_a^0 und dem ersten Routen ergibt.
$SP(u, v)$	die Länge des kürzesten Pfades von u nach v in $\mathcal{N}^*$
U_a	obere Schranke für Kante a
UB_1, UB_2	obere Schranken für das aperiodische Fahrplanproblem mit OD-Paaren
V	die Menge der Knoten in einem Graphen
w_a	Die Anzahl der Passagiere, die Aktivitat $a \in \mathcal{A}$ benutzen
w_{uv}	Die Anzahl der Passagiere, die von u nach v fahren

Inhaltsverzeichnis

Teil I

Theoretischer Teil

1 Einleitung

Die Fahrplangestaltung spielt in der heutigen Zeit eine wichtige Rolle. Da der Bestand von Ressourcen wie Öl oder Kohle immer geringer wird, der Bedarf an Energie und Kraftstoffen, zum Beispiel wegen des Wachstums der Menschheit, jedoch steigt und die Forschung der erneuerbaren Energien noch am Anfang steht, sollte für die Menschen ein Anreiz gegeben werden, verstärkt öffentliche Verkehrsmittel zu benutzen. Dies kann zum einen über den Fahrpreis, und zum anderen über eine möglichst kurze Reisezeit geschehen. Das Problem, die Reisezeit der Fahrgäste zu minimieren, hängt dabei vom Fahrplan ab, da sich die Passagiere für sich selbst eine möglichst kurze Reisezeit, also einen möglichst kurzen Weg suchen würden. Andererseits hängt der Fahrplan von der Wahl der Wege der Passagiere ab, denn wenn mehr Fahrgäste einen bestimmten Weg nehmen, ist es sinnvoll diesen Weg besonders "kurz" zu machen. Dieses Problem wird "aperiodisches Fahrplanproblem mit OD-Paaren" genannt.

Es stellt sich heraus, dass dieses Problem $\mathcal{NP}$-schwer ist und daher im Fall $\mathcal{P} \neq \mathcal{NP}$ kein Lösungsverfahren existiert, das alle Instanzen des Problems schneller als in exponentieller Zeit löst. Aus diesem Grund wird in dieser Arbeit eine Heuristik vorgestellt, die die Lösung des Problems in vielen Fällen möglichst gut annähert und in polynomieller Zeit arbeitet. Die Heuristik basiert dabei auf der Beobachtung, dass man das aperiodische Fahrplanproblem mit OD-Paaren in die beiden Teilprobleme Routing (also das Suchen kürzester Wege) und Fahrplangestaltung ohne OD-Paare unterteilen kann, welche beide in $\mathcal{P}$ liegen. Dieses Verfahren wurde in der Praxis bisher zwar schon durchgeführt, der neue Ansatz liegt jedoch darin, Routing und Fahrplangestaltung iteriert (also mehrmals hintereinander) auszuführen. Die Ergebnisse zeigen, dass sich dieser Ansatz lohnt.

Der erste Abschnitt dieser Arbeit ist der theoretische Teil. Es ist wichtig, sich zunächst mit den Grundlagen, die für das Verstehen des aperiodischen Fahrplanproblems mit OD-Paaren wichtig sind, vertraut zu machen. Dazu gehören die Komplexitätstheorie, um eine Vorstellung vom Aufwand für Algorithmen zu bekommen, sowie die Graphentheorie, die lineare Optimierung und die Verkehrsplanung, um das Problem zu modellieren. Darauf aufbauend wird die Problemstellung sowie deren Modellierung durch ganzzahlige lineare Programme beschrieben und die iterative Heuristik vorgestellt. In diesem Zusammenhang werden auch einige untere Schranken für das aperiodische Fahrplanproblem mit OD-Paaren präsentiert, sowie eine Fehlerabschätzung gegeben.

Der zweite Abschnitt dieser Arbeit ist der praktische Teil, in dem zum einen auf die Implementierung der Heuristik eingegangen wird, und zum anderen auf die durch

die Heuristik berechneten Ergebnisse, sowie deren Interpretation. Zu diesem Zweck wird für einige Instanzen versucht, mit Hilfe der ganzzahligen linearen Programme, das Problem der aperiodischen Fahrplangestaltung zu lösen. Bei den Instanzen, bei denen dies möglich ist, wird die so berechnete Lösung mit der heuristischen Lösung verglichen.

Für die Beschreibung der Grundlagen werden wir uns hauptsächlich auf das Optimierungsskript [Sch06], sowie das Verkehrsplanungsskript [Sch10] von Prof. Schöbel beziehen. Das aperiodische Fahrplanproblem mit OD-Paaren ist ein Problem, welches noch nicht lange betrachtet wird. Marie Schmidt hat sich in ihrer Dissertation [Sch11] unter anderem mit diesem Problem beschäftigt. Viele Aspekte dieser Arbeit sind in Zusammenarbeit mit ihr bearbeitet worden. Für Informationen über das periodische Fahrplanproblem mit OD-Paaren ist die Diplomarbeit von Michael Siebert [Sie11] zu empfehlen.

2 Grundlagen

2.1 Komplexitätstheorie

Um abzuschätzen, wie lange ein Algorithmus braucht, um etwas zu berechnen oder auszuführen oder wie hoch der Bedarf an Speicherplatz ist, soll zunächst eine Vorstellung davon gegeben werden, wie man diese Sachen misst. So ist es zum Beispiel nicht effizient, wenn ein Algorithmus zur Berechnung einer Funktion exponentiell viele Schritte (in Bezug auf die Problemgröße) braucht. Eine polynomielle Anzahl an Schritten hingegen wäre zu vertreten.

Da nicht die exakte Anzahl an Schritten von Interesse ist, sondern das asymptotischen Verhalten, werden zunächst die Landau-Symbole eingeführt (auch $\mathcal{O}$-Notation genannt).

Definition 2.1.1: [Bre92]
Seien $f : \mathbb{N}_0 \to \mathbb{N}_0$ und $g : \mathbb{N}_0 \to \mathbb{N}_0$ zwei Funktionen. Dann sind die Landau-Symbole definiert durch:

1. $f \in \mathcal{O}(g) \Leftrightarrow \exists c \in \mathbb{N}, c > 0 \text{ mit } f \leq c \cdot g$
2. $f \in o(g) \Leftrightarrow \forall c \in \mathbb{N}, c > 0 \text{ gilt } f \leq c \cdot g$
3. $f \in \Omega(g) \Leftrightarrow \exists c \in \mathbb{N}, c > 0 \text{ mit } f \geq c \cdot g$
4. $f \in \omega(g) \Leftrightarrow \forall c \in \mathbb{N}, c > 0 \text{ gilt } f \geq c \cdot g$
5. $f \in \Theta(g) \Leftrightarrow f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f)$

Häufig schätzt man die Laufzeit eines Algorithmus ab, indem man ihn in Teilprobleme spaltet und die Laufzeit dieser Teilprobleme berechnet. Um die Gesamtlaufzeit zu berechnen, sind folgende Rechenregeln für die Landau-Symbole hilfreich:

Satz 2.1.2: [Bre92]
Seien $f : \mathbb{N}_0 \to \mathbb{N}_0$ und $f : \mathbb{N}_0 \to \mathbb{N}_0$ Funktionen und $k \in \mathbb{N}_0$ eine Konstante. Dann gelten:

1. $f(n) \in \mathcal{O}(f(n))$
2. $k \cdot \mathcal{O}(f(n)) \in \mathcal{O}(f(n))$
3. $\mathcal{O}(f(n)) + \mathcal{O}(f(n)) \in \mathcal{O}(f(n))$
4. $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) \in \mathcal{O}(f(n) \cdot g(n))$

5. $\mathcal{O}(\mathcal{O}(f(n))) \in \mathcal{O}(f(n))$

Bemerkung 2.1.3:
Anstatt $f \in \mathcal{O}(g)$ für zwei Funktionen f und g schreibt man auch häufig $f = \mathcal{O}(g)$. Das "=" ist dabei aber kein $=$ im mathematischen Sinne.

Definition 2.1.4: [CLRS07]
Eine Instanz I eines Problems sind alle Eingaben, die die vom Problem vorgegebenen Bedingungen erfüllen und die gebraucht werden, um eine Lösung des Problems zu berechnen.

Wir definieren nun, wann ein Problem polynomiell lösbar ist:

Definition 2.1.5: [Sch10]
Ein Problem P heißt polynomiell lösbar, wenn es ein festes $q \in \mathbb{N}$ und einen Algorithmus gibt, der jede Instanz von P von Größe n in einer Zeit von $\mathcal{O}(n^q)$ löst.

Definition 2.1.6: [CLRS07]
Ein Entscheidungsproblem ist ein Problem, bei dem die Antwort entweder ja oder nein ist.

Definition 2.1.7: [Sch10]
Seien A und B zwei Probleme. Dann heißt A polynomiell reduzierbar auf B (in Zeichen $A \propto B$), wenn jede Instanz von A in polynomieller Zeit (also in $\mathcal{O}(n^q)$ für festes q und n als Größe der Instanz) korrekt in B überführt werden kann.

Wir werden nun die Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$, sowie die Menge der $\mathcal{NP}$-harten und $\mathcal{NP}$-vollständigen Probleme kennenlernen.

Definition 2.1.8: [Sch10],[Bre92]

1. $\mathcal{NP}$ ist die Menge der Probleme, für die eine geratene Lösung in polynomieller Zeit überprüft werden kann.

2. $\mathcal{P}$ ist die Menge aller Probleme, die in polynomieller Zeit gelöst werden können.

3. $\mathcal{NPH} = \{B \mid \forall A \in \mathcal{NP} : A \propto B\}$ ist die Menge der $\mathcal{NP}$-schweren ($\mathcal{NP}$-harten) Probleme.

4. $\mathcal{NPC} = \{B \in \mathcal{NP} \mid \forall A \in \mathcal{NP} : A \propto B\}$ ist die Menge der $\mathcal{NP}$-vollständigen Probleme.

Der Unterschied zwischen $\mathcal{NP}$-schweren und $\mathcal{NP}$-vollständigen Problemen besteht im Definitionsbereich. Alle $\mathcal{NP}$-vollständigen Probleme sind $\mathcal{NP}$-schwer, aber nicht alle $\mathcal{NP}$-schweren Probleme sind $\mathcal{NP}$-vollständig.

Es ist bekannt, dass $\mathcal{P} \subset \mathcal{NP}$. Ob auch $\mathcal{NP} \subset \mathcal{P}$ gilt, ist noch nicht beantwortet. Es wird jedoch davon ausgegangen, dass $\mathcal{NP} \not\subset \mathcal{P}$ gilt.

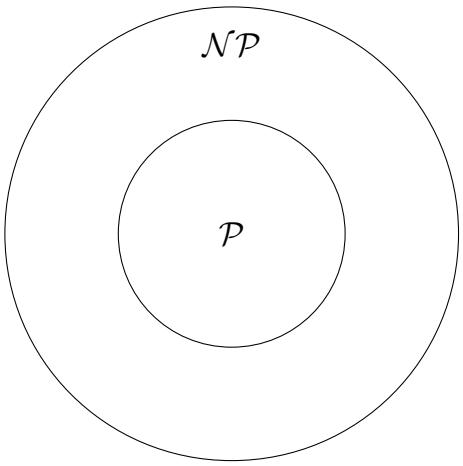


Abbildung 2.1: $\mathcal{P} \subset \mathcal{NP}$

2.2 Graphentheorie

Da das aperiodische Fahrplanproblem mit OD-Paaren als Netzwerk modelliert wird, werden zunächst einige Begriffe aus der Graphentheorie eingeführt.

Definition 2.2.1: [Fou09]
Ein ungerichteter Graph $G = (V, E)$ besteht aus einer nichtleeren, endlichen Menge V von Elementen, die Knoten genannt werden, und einer endlichen Menge E von Elementen, genannt Kanten. Jede Kante $e \in E$ verbindet jeweils zwei Knoten i und j aus V. Dabei sind $i \in V$ und $j \in V$ die Endknoten von e.

Notation 2.2.2:
Eine ungerichtete Kante $e \in E$, die die Knoten i und j verbindet, schreiben wir als $e = \{i, j\}$.

Definition 2.2.3: [Fou09]
Ein gerichteter Graph $G = (V, A)$ besteht aus einer nichtleeren, endlichen Menge V von Knoten und einer endlichen Menge A von Pfeilen oder gerichteten Kanten, die jeweils einen Knoten $i \in V$ mit einem Knoten $j \in V$ verbinden.

Wenn nicht erläutert wird, ob der Graph gerichtet oder ungerichtet ist, treffen die Aussagen sowohl für gerichtete als auch für ungerichtete Graphen zu.

Notation 2.2.4:
Eine gerichtete Kante $a \in A$ von $i \in V$ nach $j \in V$ schreiben wir als $a = (i, j)$.

Definition 2.2.5: [KNW05]
Sei $G = (V, A)$ ein gerichteter Graph. $\alpha : A \to V$ und $\omega : A \to V$ seien Abbildungen. Für $a = (i, j)$ heißt $\alpha(a) = i$ Anfangsknoten und $\omega(a) = j$ Endknoten von a.

Definition 2.2.6: [KNW05]
Sei $G = (V, E)$ ein Graph. Eine Kante $e \in E$ heißt Schlinge, wenn $\alpha(e) = \omega(e)$. Zwei

Kanten $e_1, e_2 \in E, e_1 \neq e_2$ heißen parallel, wenn $\alpha(e_1) = \alpha(e_2)$ und $\omega(e_1) = \omega(e_2)$.

Definition 2.2.7: [KNW05]
Ein Graph $G = (V, E)$ heißt einfach, wenn er keine Schlingen und parallelen Kanten enthält.

Wenn keine anderen Angaben gemacht werden, sind die Graphen, die wir betrachten, alle einfach.

Definition 2.2.8: [Bol90]

- Sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V', E')$ mit $V' \subset V$ und $E' \subset E$ wird Subgraph von G genannt.
- Wenn G' alle Kanten aus G enthält, die zwei Knoten in V' verbinden, heißt G' der von V' induzierte Subgraph und wird mit $G[V']$ bezeichnet.

Definition 2.2.9: [KNW05]

- Ein Weg P im Graph $G = (V, E)$ ist eine endliche Folge von Knoten und Kanten $P = (v_0, e_1, v_1, e_2, \ldots, v_n)$ wobei $v_i \in V \; \forall i \in 0, \ldots, n$ und $e_i = (v_{i-1}, v_i) \in E \; \forall i = 1, \ldots, n$.
- Falls $v_0 = v_n$ ist P ein Kreis.
- Ein Weg P wird Pfad genannt, falls für alle $i, j \in \{1, \ldots, n\}$ mit $i \neq j$ gilt $v_i \neq v_j$.

Bemerkung 2.2.10:
Da die Graphen, die wir betrachten, einfach sind, schreiben wir einen Weg P auch durch eine Folge von Knoten $P = (v_0, v_1, \ldots, v_n)$.

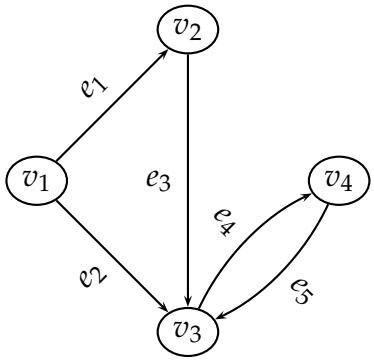


Abbildung 2.2: Einfacher Graph. e_4 und e_5 sind nicht parallel, da $\alpha(e_4) = v_3 \neq v_4 = \alpha(e_5)$

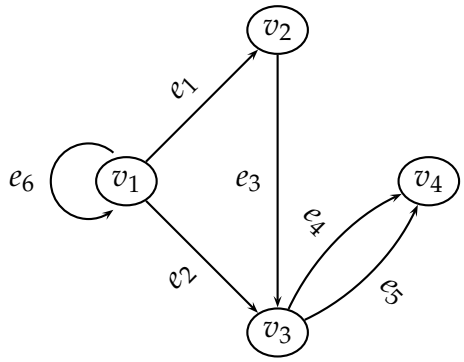


Abbildung 2.3: Graph mit Schlinge e_6 und parallelen Kanten e_4 und e_5

Definition 2.2.11: [Fou09]
Sei $G = (V, E)$ ein Graph.

- G heißt zusammenhängend, wenn es für jedes Paar $\{i, j\}$ von unterschiedlichen Knoten einen Weg von i nach j gibt.
- $G' = (V', E')$ ist ein maximal zusammenhängender Subgraph, wenn G' zusammenhängend ist und jeder andere G' enthaltende Subgraph von G nicht zusammenhängend ist.
- Die Zusammenhangskomponenten von G sind die maximal zusammenhängend induzierten Subgraphen von G.

Definition 2.2.12: [KNW05]
Sei $G = (V, E)$ ein gerichteter Graph.

- $\delta^+(v) = \{e \in E \mid \alpha(e) = v\}$ ist die Menge der von v ausgehenden Kanten.
- $\delta^-(v) = \{e \in E \mid \omega(e) = v\}$ ist die Menge der in v eingehenden Kanten.
- $d^+(v) = |\delta^+(v)|$ ist der Außengrad von v.
- $d^-(v) = |\delta^-(v)|$ ist der Innengrad von v.

Da die Heuristik einen Kürzeste-Wege-Algorithmus benutzt, werden zwei derartige vorgestellt. Der erste Algorithmus (Dijkstra) kann nur kürzeste Wege in Graphen mit positiven Kantengewichten für alle Kanten berechnen.

Algorithmus 2.2.13: Dijkstra [HK06]
Input: Gerichteter Graph $G = (V, E)$ mit Gewichten $c(e) \geq 0 \; \forall e \in E$, Startknoten s
Output: Kürzeste Wege von s zu allen Knoten

1: Setze $d_i := \infty \; \forall i = 1, \ldots n$, $d_s := 0$, $d(s) := 0$, $p := 1$
2: Setze $X_p := s$
3: $\rho := \begin{cases} c_{si}(s,i) \in E \\ \infty, \text{sonst} \end{cases} \quad \forall i \in V \setminus X_p$
4: $pred(i) = s \; \forall i \in V \setminus X_p$
5: Bestimme i_{p+1} mit $\rho_{i_{p+1}} = \min\limits_{i \in V \setminus X_p} \rho_i$ und setze $d_{i_{p+1}} := \rho_{i_{p+1}}$
6: **if** $\rho_{i_{p+1}} = \infty$ **then**
7: STOP, es gibt keinen Weg von s nach $i \; \forall i \in V \setminus X_p$
8: **end if**
9: Setze $X_{p+1} := X_p \cup \{i_{p+1}\}$, $p := p + 1$
10: **if** $p = n$ **then**
11: STOP: Kürzeste Wege sind gefunden
12: **end if**
13: **for** $i \in V \setminus X_p$ **do**
14: **if** $\rho_i > d_{i_p} + c_{i_p,i}$ **then**
15: $\rho_i = d_{i_p} + c_{i_p,i}$
16: $pred(i) := i_p$
17: **end if**
18: **end for**
19: Goto 5.

Wir werden außerdem einen Algorithmus brauchen, der negative Kreise erkennt. Dazu bietet sich der Bellman-Ford-Algorithmus (auch Label Correcting Algorithmus genannt) an. Dieser lässt als Eingabe auch negative Kantengewichte zu.

Algorithmus 2.2.14: [KNW05]
Input: gerichteter Graph $G = (V, A)$, Gewichtsfunktion $c : A \to \mathbb{R}$, Knoten s
Output: Baum kürzester Wege von s

```
1: for v ∈ V do
2:    d[v] := +∞
3:    Π[v] := NIL
4: end for
5: d[s] := 0
6: for i = 1, ..., |V| − 1 do
7:    for (u, v) ∈ A do
8:       if d[v] > d[u] + c(u, v) then
9:          d[v] = d[u] + c(u, v)
10:         Π[v] = u
11:      end if
12:   end for
13: end for
14: for (u, v) ∈ A do
15:   if d[v] > d[u] + c(u, v) then
16:      return "Ein negativer Kreis ist von s erreichbar."
17:   end if
18: end for
```

2.3 Lineare Optimierung

Wir werden später ein ganzzahliges lineares Programm einführen, das das aperiodische Fahrplanproblem mit OD-Paaren löst. Deshalb brauchen wir einige Grundlagen aus dem Bereich der linearen Optimierung.

In der linearen Optimierung werden lineare Funktionen unter bestimmten linearen Nebenbedingungen minimiert beziehungsweise maximiert. Im Folgenden wird zunächst die Gestalt einiger linearer Programme beschrieben.

Definition 2.3.1: [Sch07]
Ein allgemeines lineares Programm hat die Form:

$$
\begin{aligned}
&\min\ c^T x \\
&\text{s.d. } Ax = d && && (1)\\
&\qquad Bx \leq e && && (2)\\
&\qquad Cx \geq f && && (3)\\
&\qquad x_i \geq 0 && \forall i = 1, \ldots, n_1 && (4)\\
&\qquad x_i \leq 0 && \forall i = n_1 + 1, \ldots, n_2 && (5)\\
&\qquad x_i \lesseqgtr 0 && \forall i = n_2 + 1, \ldots, n. && (6)
\end{aligned}
$$

Dabei sind $c, x \in \mathbb{R}^n, d \in \mathbb{R}^k, e \in \mathbb{R}^l, f \in \mathbb{R}^m, A \in \mathbb{R}^{k \times n}, B \in \mathbb{R}^{l \times n}, C \in \mathbb{R}^{m \times n}$.

Definition 2.3.2: [Sch07]

- $x \in \mathbb{R}^n$ heißt zulässige Lösung, falls x die Nebenbedingungen (1)-(6) erfüllt.
- Eine zulässige Lösung $x \in \mathbb{R}^n$ heißt Optimallösung, falls $c^T x \leq c^T y$ für alle zulässigen Lösungen $y \in \mathbb{R}^n$.

Wir suchen in einem allgemeinen linearen Programm der Form von Definition 2.3.1 also ein x, für das $c^T x$ minimal wird und das die Nebenbedingungen (1)-(6) erfüllt.

Notation 2.3.3: [FP93]
Ein lineares Programm in Standardform wird folgendermaßen geschrieben:

$$\begin{array}{ll} \min & c^T x \\ \text{s.d.} & Ax = b \\ & x \in \mathbb{R}^n_+ \end{array}$$

mit $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$.

Notation 2.3.4: [Sch07]
Ein lineares Programm

$$\begin{array}{ll} \min & c^T x \\ \text{s.d.} & Ax = b \\ & x \in \mathbb{R}^n_+ \end{array}$$

schreiben wir auch $\min\{c^T x \mid Ax = b, x \in \mathbb{R}^n\}$

Satz 2.3.5: [HK06]
Jedes lineare Programm kann in ein lineares Programm in Standardform überführt werden.

Definition 2.3.6: [HK06]
Sei

$$(P) \qquad \begin{array}{ll} \min & c^T x \\ \text{s.d.} & Ax = b \\ & x \in \mathbb{R}^n_+. \end{array}$$

mit $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$ ein lineares Programm in Standardform (auch primales lineares Programm genannt). Das zu (P) duale Programm ist

$$(D) \qquad \begin{array}{ll} \max & b^T \pi \\ \text{s.d.} & A^T \pi^T = c^T \\ & \pi \lneqq 0. \end{array}$$

Da sich nach Satz 2.3.5 jedes lineare Programm in ein Programm in Standardform überführen lässt, kann man jedes lineare Programm dualisieren. Der folgende Satz stellt den Zusammenhang zwischen primalen und dualen linearen Programmen her.

Satz 2.3.7: [HK06]
Sei (P) ein allgemeines lineares Programm. Dann gilt:

1. Ist x zulässig für (P) und π zulässig für (D), so gilt
$$b^T \pi^T \leq cx.$$
2. Hat entweder (P) oder (D) eine endliche Optimallösung, so auch das andere und die optimalen Zielfunktionswerte sind gleich.
3. Ist entweder (P) oder (D) unbeschränkt, so ist das dazu duale Programm unzulässig.
4. (P) und (D) können unzulässig sein.

1. heißt auch "Schwacher Dualitätssatz" und 2.-4. "Starker Dualitätssatz"

Bemerkung 2.3.8:
Wenn alle Variablen eines linearen Programms (P) ganzzahlig sind, so nennt man (P) ein ganzzahliges (lineares) Programm oder auch integer linear program (Abkürzung: ILP).

Definition 2.3.9: [Sch07]
Sei
$$\begin{array}{lll} (P) & \min & c^T x \\ & \text{s.d.} & Ax = b \\ & & x \in \mathbb{Z}^n_+ \end{array}$$
ein ganzzahliges lineares Programm. Die LP-Relaxation von (P) ist
$$\begin{array}{lll} (P_{relax}) & \min & c^T x \\ & \text{s.d.} & Ax = b \\ & & x \in \mathbb{R}^n_+ \end{array}$$

Satz 2.3.10:
Sei (P) $\min\{c^T x \mid Ax = b, x \in \mathbb{Z}\}$ ein ganzzahliges lineares Programm und (P_{relax}) $\min\{c^T x \mid Ax = b, x \in \mathbb{R}\}$ die LP-Relaxation von (P). Seien weiterhin z^* der Zielfunktionswert von (P) und z^{relax} der Zielfunktionswert von P_{relax}. Dann gilt:
$$z^{relax} \leq z^* \ .$$

Satz 2.3.11: [Sch07]
Sei
$$\begin{array}{lll} (P) & \min & c^T x \\ & \text{s.d.} & Ax = b \\ & & x \in \mathbb{Z}^n_+ \end{array}$$

ein ganzzahliges lineares Programm und

$$\begin{array}{lll} (P_{relax}) & \min & c^T x \\ & \text{s.d.} & Ax = b \\ & & x \in \mathbb{R}^n_+ \end{array}$$

die LP-Relaxation von (P). Seien weiterhin x' optimal für P und x^* optimal für P_{relax}. Dann gelten

1. $c^T x^* \leq c^T x'$
2. Falls $x^* \in \mathbb{Z}^n$ und $c^T x^* = c^T x'$, dann ist x^* optimal für (P).

Der Zielfunktionswert der LP-Relaxation ist im Fall eines Minimierungsproblems eine untere Schranke für den Zielfunktionswert des ganzzahligen Programms und wenn die Lösung der LP-Relaxation ganzzahlig ist, so ist sie Optimallösung für das ganzzahlige Programm.

Definition 2.3.12: [Sch07]
Sei $A \in \mathbb{K}^{m \times n}$ eine Matrix. Eine Submatrix oder Untermatrix A' von A entsteht dann durch Streichen von Zeilen und Spalten von A.

Definition 2.3.13: [Sch07]
Sei $A \in \mathbb{K}^{m \times n}$. A heißt total unimodular, falls jede Untermatrix A' von A Determinante ± 1 oder 0 hat.

Lemma 2.3.14: [Sch07]
Sei $A \in \mathbb{K}^{m \times n}$.

1. A ist genau dann total unimodular, wenn A^T total unimodular ist.
2. A ist genau dann total unimodular, wenn die um die Einheitsmatrix $I \in \mathbb{K}^{m \times m}$ erweiterte Matrix $(A \mid I)$ total unimodular ist.

Lemma 2.3.15: [Sch07]
Sei $A \in \mathbb{K}^{m \times n}$ ganzzahlig und $\min\{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$ ein ganzzahliges lineares Programm. Dann sind äquivalent:

1. A ist total unimodular.
2. Für alle $c \in \mathbb{R}^n, b \in \mathbb{Z}^m$ gilt
$$\min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n_+\} = \min\{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$$

2.4 Verkehrsplanung

Das Problem der aperiodischen Fahrplangestaltung wird mit Hilfe eines sogenannten Ereignis-Aktivitätsnetzwerkes modelliert. Da das Ereignis-Aktivitätsnetzwerk auf einem öffentlichen Verkehrsnetzwerk basiert, werden zunächst einige Begriffe in diesem Zusammenhang vorgestellt. Später werden wir davon ausgehen, dass das Ereignis-Aktivitätsnetzwerk vorgegeben ist.

2.4.1 Grundlagen

Definition 2.4.1: [Sch06]
Ein öffentliches Verkehrsnetzwerk (Public Transportation Network, PTN) ist ein ungerichteter Graph $PTN = (V, E)$, der durch eine Menge an Haltestellen (oder Bahnhöfen) V und eine Menge E an direkten Verbindungen zwischen den Haltestellen gegeben ist.

Die Aussage, dass das PTN ungerichtet ist, kann man sich so vorstellen, dass es keine "Einbahnstraßen" gibt. Man kann also von einem Bahnhof i zu einem anderen Bahnhof j und auch wieder zurück gelangen.

Definition 2.4.2: [Sch06]

- Eine Linie ist ein Weg im öffentlichen Verkehrsnetzwerk.
- Die Frequenz f_l einer Linie l gibt an, wie oft Linie l innerhalb der gegebenen Zeit T gefahren wird.

Definition 2.4.3: [Sch06]
Die Menge aller Linien $\mathcal{L}$ zusammen mit den Frequenzen $f_l \in \mathbb{N}\ \forall l \in \mathcal{L}$, wie oft die Linie gefahren wird, heißt Linienplan $(\mathcal{L}, f_l)$.

2.4.2 Ereignis-Aktivitätsnetzwerk

Definition 2.4.4: [Sch06]

1. Ein Ankunftsereignis ist ein Tripel (v, l, arr), das die Ankunft eines Zuges der Linie l an Bahnhof v angibt. Analog gibt ein Abfahrtsereignis (v, l, dep) die Abfahrt eines Zuges der Linie l von Bahnhof v an.

2. Die Menge der Ankunfts- und Abfahrtsereignisse sind gegeben durch

$$\mathcal{E}_{arr} = \{(v, l, arr) : v \in V, l \in \mathcal{L}\}$$
$$\mathcal{E}_{dep} = \{(v, l, dep) : v \in V, l \in \mathcal{L}\}$$

Beispiel 1:
Seien l_1, l_2 zwei Linien und Göttingen ein Bahnhof.

- $(\text{Göttingen}, l_1, arr)$ ist ein Ankunftsereignis eines Zuges der Linie l_1 im Bahnhof Göttingen
- $(\text{Göttingen}, l_2, dep)$ ist ein Abfahrtsereignis eines Zuges der Linie l_2 im Bahnhof Göttingen

Die Ereignisse werden die Knoten darstellen. Weiterhin brauchen wir Kanten zwischen je zwei Ereignissen. Diese Kanten heißen Aktivitäten. Um die Notation zu ver-

einfachen, führen wir nun sogenannte Ereignis-Aktivitätsnetzwerke ein. Ein Ereignis-Aktivitätsnetzwerk besteht aus Abfahrts- und Ankunftsereignissen und aus Umsteige-, Fahr- und Warteaktivitäten. Formal:

Definition 2.4.5: [Sch06]
Sei $PTN = (V, E)$ ein öffentliches Verkehrsnetzwerk mit einer Menge an Ereignissen $\mathcal{E}_{arr}$ und $\mathcal{E}_{dep}$ und eine Menge von erforderlichen Umstiegen. Dann ist das Ereignis-Aktivitätsnetzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ gegeben durch

$$\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$$
$$\mathcal{A} = \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}_{change}$$

wobei

$$\mathcal{A}_{drive} = \{((v_1, l, dep), (v_2, l, arr)) \in \mathcal{E}_{dep} \times \mathcal{E}_{arr}\}$$
$$\mathcal{A}_{wait} = \{((v, l, arr), (v, l, dep)) \in \mathcal{E}_{arr} \times \mathcal{E}_{dep}\}$$
$$\mathcal{A}_{change} = \{((v, l_1, arr), (v, l_2, dep)) \in \mathcal{E}_{arr} \times \mathcal{E}_{dep}\}$$

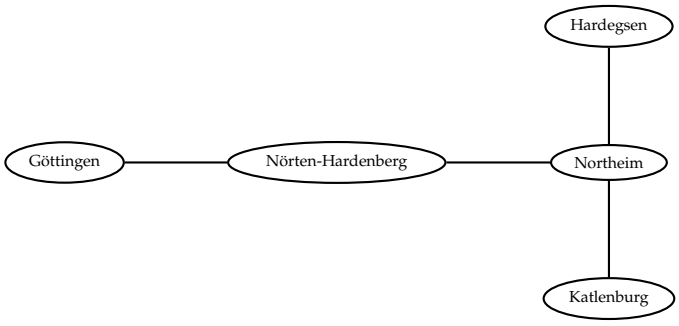
Wichtig ist:

- Eine Fahraktivität $a \in \mathcal{A}_{drive}$ verbindet ein Abfahrtsereignis mit einem Ankunftsereignis an unterschiedlichen Bahnhöfen mit gleicher Linie.
- Eine Warteaktivität $a \in \mathcal{A}_{wait}$ verbindet ein Ankunftsereignis mit einem Abfahrtsereignis am gleichen Bahnhof mit gleicher Linie.
- Eine Umsteigeaktivität $a \in \mathcal{A}_{change}$ verbindet ein Ankunftsereignis mit einem Abfahrsereignis am gleichen Bahnhof mit unterschiedlichen Linien.

Wir werden den Aktivitäten später noch "Gewichte" in Form von unteren und oberen Schranken für die Fahr-, Warte-, beziehungsweise Umsteigezeit zuordnen.

Beispiel 2 zeigt ein öffentliches Verkehrsnetzwerk PTN und ein zu PTN zugehöriges Ereignis-Aktivitätsnetzwerk. Wie man in diesem Beispiel sieht, kann schon ein kleines öffentliches Verkehrsnetzwerk ein Ereignis-Aktivitätsnetzwerk mit vielen Knoten erzeugen.

Beispiel 2:



Wir benutzen nun folgende Abkürzungen: Göttingen - *Gö*, Nörten-Hardenberg - *NH*, Northeim - *Nom*, Hardegsen - *Hg*, Katlenburg - *Kb*. Angenommen wir hätten nun folgende Linien:

$$l_1 = (Gö, NH, Nom, Kb)$$

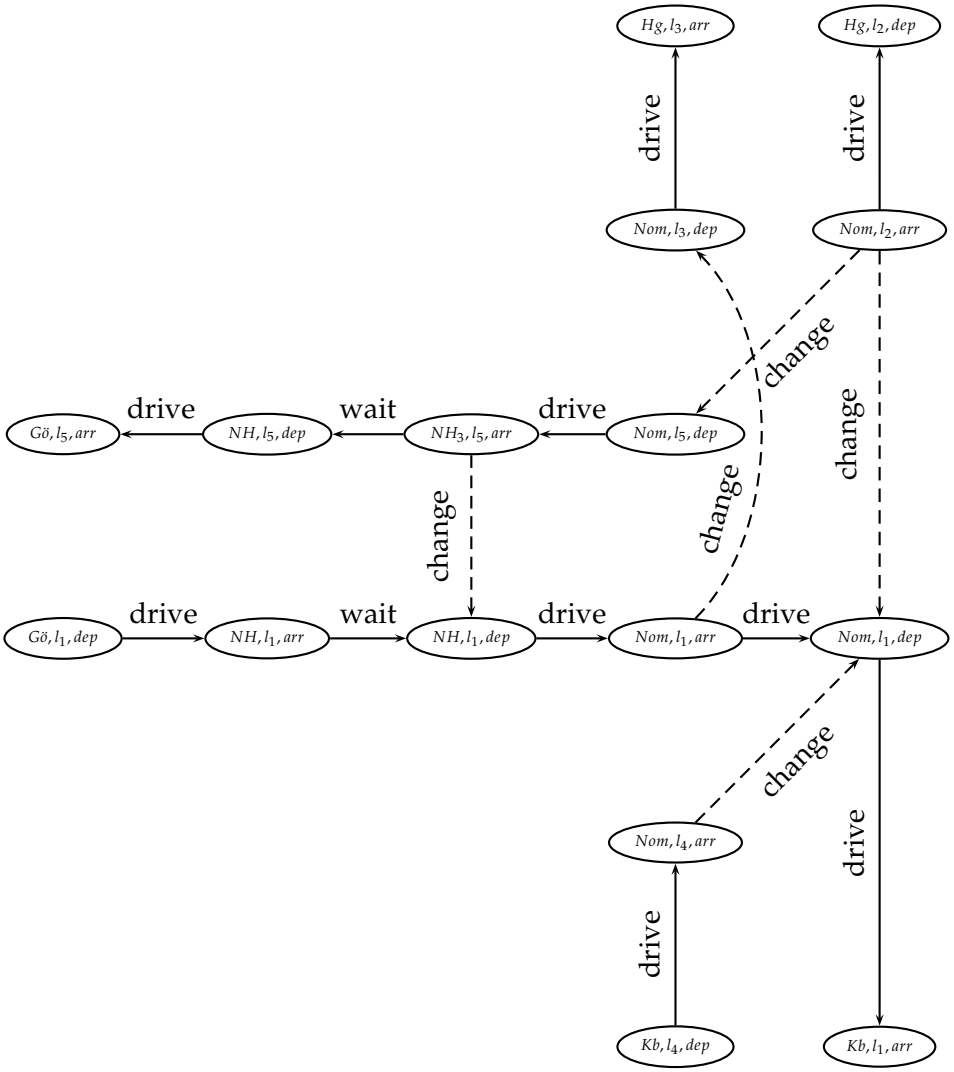
$$l_2 = (Nom, Hg)$$

$$l_3 = (Nom, Hg)$$

$$l_4 = (Kb, Nom)$$

$$l_5 = (Nom, NH, Gö)$$

Dann könnte das Ereignis-Aktivitätsnetzwerk dazu folgendermaßen aussehen:



Wir gehen im Folgenden davon aus, dass wir die Menge der Abfahrts- und Ankunftsereignisse schon gefunden haben und wollen nun definieren, was ein Fahrplan ist.

Definition 2.4.6: [Sch06]
Ein Fahrplan ist gegeben durch natürliche Zahlen

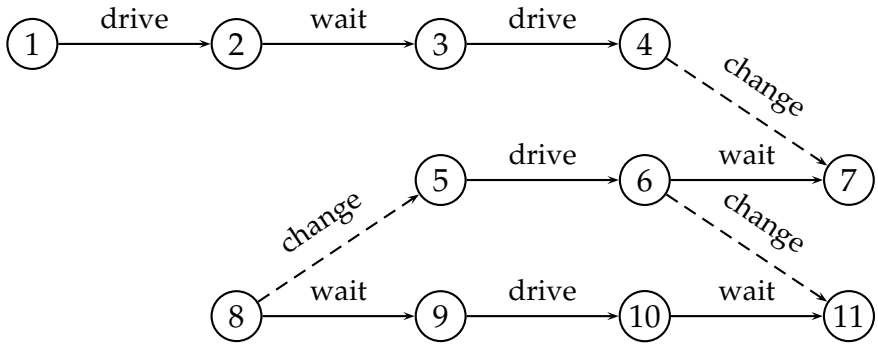
- $\Pi_{v,l,arr}$ für alle $(v, l, arr) \in \mathcal{E}_{arr}$

und

- $\Pi_{v,l,dep}$ für alle $(v, l, dep) \in \mathcal{E}_{arr}$

Bemerkung 2.4.7:
Da ein Ereignis $(v, l, dep) \in \mathcal{E}_{arr}$ beziehungsweise $(v, l, arr) \in \mathcal{E}_{dep}$ eindeutig bestimmt ist, schreiben wir kurz $i \in \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$ und damit auch $\Pi_i \in \mathcal{N}$ für alle $i \in \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$.

Beispiel 3:



Ein möglicher Fahrplan wäre dann:

$$\begin{aligned}
\Pi_1 &= \Pi_{(v_1,l_1,dep)} &&= 1\\
\Pi_2 &= \Pi_{(v_2,l_1,arr)} &&= 10\\
\Pi_3 &= \Pi_{(v_2,l_1,dep)} &&= 11\\
\Pi_4 &= \Pi_{(v_3,l_1,arr)} &&= 20\\
\Pi_5 &= \Pi_{(v_4,l_2,dep)} &&= 1\\
\Pi_6 &= \Pi_{(v_3,l_2,arr)} &&= 24\\
\Pi_7 &= \Pi_{(v_3,l_2,dep)} &&= 25\\
\Pi_8 &= \Pi_{(v_4,l_3,arr)} &&= 0\\
\Pi_9 &= \Pi_{(v_4,l_3,dep)} &&= 10\\
\Pi_{10} &= \Pi_{(v_3,l_3,arr)} &&= 15\\
\Pi_{11} &= \Pi_{(v_3,l_3,dep)} &&= 28
\end{aligned}$$

2.4.3 Aperiodische Fahrplangestaltung ohne OD-Paare

Wir lernen nun kennen, wann ein Fahrplan aperiodisch zulässig ist und wollen untersuchen, wie man einen zulässigen Fahrplan berechnen kann. Wir geben dazu allen Aktivitäten im Ereignis-Aktivitätsnetzwerk untere und obere Schranken, die die

minimale und maximale Dauer der Aktivitäten beschreiben. Außerdem wollen wir versuchen, einen möglichst guten zulässigen Fahrplan zu bestimmen. Zunächst nehmen wir dafür an, dass die Anzahl der Passagiere w_a, die Aktivität $a \in A$ "benutzen", schon bekannt ist.

Definition 2.4.8:
Sei $w_a : \mathcal{A} \to \mathbb{N}$ eine Abbildung. w_a gibt die Anzahl der Passagiere an, die Aktivität $a \in \mathcal{A}$ benutzen und wird auch Passagierverteilung genannt.

Notation 2.4.9: [Sch10]
Für jede Aktivität $a \in \mathcal{A}$ bezeichne $0 \leq L_a \leq U_a$ die minimal und maximal erlaubte Dauer der Aktivität a. $\Delta_a = [L_a, U_a]$ heißt Span oder Zeitspanne von a.

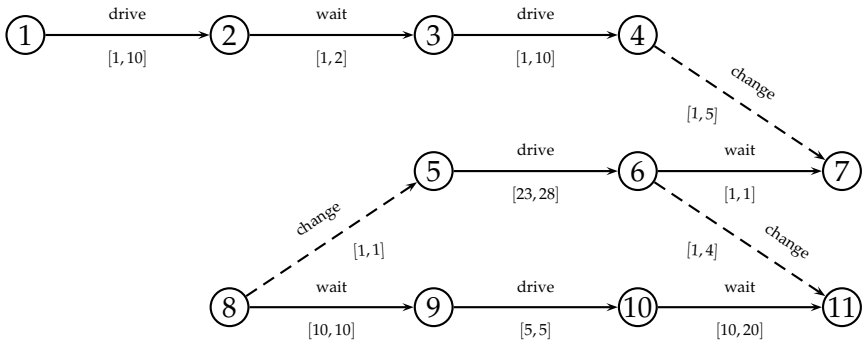
Nach [Sch10] kann die untere Schranke für eine Fahraktivität $a \in \mathcal{A}_{drive}$ zum Beispiel die Zeit angeben, die man braucht um von einem Bahnhof zu einem anderen zu gelangen, wenn man mit höchstmöglicher Geschwindigkeit fährt. Die obere Schranke gibt an, wieviel Zeit man den Kunden für diese Aktivität zumuten kann.
Für $a = ((v, t, arr), (v, t, dep)) \in \mathcal{A}_{wait}$ beinhaltet die untere Schranke L_a Faktoren wie nötige Pausen, mögliche Fahrerwechsel oder Wartungsarbeiten.
Für $a = ((v, l_1, arr), (v, l_2, dep)) \in \mathcal{A}_{change}$ bezeichnet die untere Schranke L_a die Zeit, die die Fahrgäste minimal brauchen, um von einem Zug der Linie l_1 in einen anderen der Linie l_2 umzusteigen (Zeit, um zu einem anderen Gleis zu gelangen). Die obere Schranke U_a bezeichnet die Zeit, die die Fahrgäste maximal bereit sind auf eine Verbindung zu warten.

Nun wollen wir definieren, wann ein Fahrplan zulässig ist.

Definition 2.4.10: [Sch10]
Ein Fahrplan Π heißt aperiodisch zulässig, wenn $\Pi_j - \Pi_i \in \Delta_a$ für alle $a = (i, j) \in \mathcal{A}$

Bemerkung 2.4.11:
Wenn nichts anderes angegeben ist, bedeutet zulässig in dieser Arbeit immer aperiodisch zulässig.

Beispiel 4:
Wir betrachten wiederum das vorherige Beispiel, ordnen den Aktivitäten jetzt aber noch untere und obere Schranken zu und nennen die Knoten der Einfachheit halber um



Der Fahrplan Π mit

$$\begin{array}{llll} \Pi_1 & = 1 & \Pi_2 & = 10 \\ \Pi_3 & = 11 & \Pi_4 & = 20 \\ \Pi_5 & = 1 & \Pi_6 & = 24 \\ \Pi_7 & = 25 & \Pi_8 & = 0 \\ \Pi_9 & = 10 & \Pi_{10} & = 15 \\ \Pi_{11} & = 28 & & \end{array}$$

ist zulässig, da

$$\begin{array}{llll} \Pi_2 - \Pi_1 & = 9 \in [1,10] & \Pi_3 - \Pi_2 & = 1 \in [1,2] \\ \Pi_4 - \Pi_3 & = 9 \in [1,10] & \Pi_6 - \Pi_5 & = 23 \in [23,28] \\ \Pi_9 - \Pi_8 & = 10 \in [10,10] & \Pi_{10} - \Pi_9 & = 5 \in [5,5] \\ \Pi_{11} - \Pi_{10} & = 13 \in [10,20] & \Pi_7 - \Pi_6 & = 1 \in [1,1] \\ \Pi_7 - \Pi_4 & = 5 \in [1,5] & \Pi_5 - \Pi_8 & = 1 \in [1,1] \\ \Pi_{11} - \Pi_6 & = 4 \in [1,4] & & \end{array}$$

Π mit

$$\begin{array}{llll} \Pi_1 & = 1 & \Pi_2 & = 10 \\ \Pi_3 & = 11 & \Pi_4 & = 20 \\ \Pi_5 & = 1 & \Pi_6 & = 24 \\ \Pi_7 & = 25 & \Pi_8 & = 0 \\ \Pi_9 & = 10 & \Pi_{10} & = 15 \\ \mathbf{\Pi_{11}} & \mathbf{= 29} & & \end{array}$$

ist hingegen nicht zulässig, da

$$\Pi_{11} - \Pi_6 = 5 \notin [1,4]$$

Es ergibt sich nun die Frage, wie man einen zulässigen Fahrplan finden kann. Wir befassen uns also mit folgendem Problem:

Problem 2.4.12: [Sch10]
Gegeben sei ein Ereignis-Aktivitätsnetzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Zeitspannen Δ_a für alle $a \in \mathcal{A}$. Zu finden ist ein zulässiger Fahrplan Π, also Werte Π_i für alle $i \in \mathcal{E}$, so dass $\Pi_j - \Pi_i \in \Delta_a$ für alle $a = (i, j) \in \mathcal{A}$

Wir führen einige Bezeichnungen ein:

Notation 2.4.13: [Sch10]
Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitätsnetzwerk und Π ein Fahrplan. Dann wird Π_i Potential in Knoten $i \in \mathcal{E}$ genannt. $\xi_a = \Pi_j - \Pi_i$ heißt Spannung, Länge oder Dauer von Aktivität $a = (i, j) \in \mathcal{A}$ bezüglich Π.

Notation 2.4.14: [Sch10]
Sei $G = (V, E)$ ein gerichteteter Graph und $C = (i_1, \ldots, i_n)$ eine Folge von Knoten.

- Sind i_j und i_{j+1} durch eine Kante verbunden, so nennt man C einen Weg.
- Ein Weg C heißt Pfad, falls alle Knoten aus C voneinander verschieden sind.

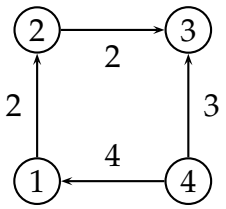
Notation 2.4.15: [Sch10]
Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitätsnetzwerk.

- Ein Pfad $C = (i_0, i_1, \ldots, i_n)$ mit $i_0 = i_n$, der $i_j \in \mathcal{E}$ erfüllt und bei dem entweder $(i_j, i_{j+1}) \in \mathcal{A}$ oder $(i_{j+1}, i_j) \in \mathcal{A}$ für alle $j = 0, \ldots, n-1$, wird ungerichteter Kreis genannt.
- Wenn $(i_j, i_{j+1}) \in \mathcal{A}$ für alle $j = 0, \ldots, n-1$, ist der ungerichtete Kreis ein Kreis.
- $C^+ = \{(i_j, i_{j+1}) : (i_j, i_{j+1}) \in \mathcal{A}\}$ heißt die Menge der Vorwärtskanten von C.
  $C^- = \{(i_{j+1}, i_j) : (i_j, i_{j+1}) \notin \mathcal{A}\}$ heißt die Menge der Rückwärtskanten von C.
- Die Spannung von C ist definiert durch

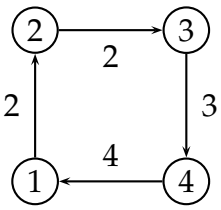
$$tension(C) = \sum_{a \in C^+} \xi_a - \sum_{a \in C^-} \xi_a \, .$$

Beispiel 5:
Wir betrachten

- den ungerichteten Kreis C_1



- und den Kreis C_2



Dann gilt $tension(C_1) = 2 + 2 - 3 + 4 = 5$ und $tension(C_2) = 2 + 2 + 3 + 4 = 11$.

Zurück zu den Ereignis-Aktivitätsnetzwerken. Eine wichtige Eigenschaft beschreibt folgendes Lemma:

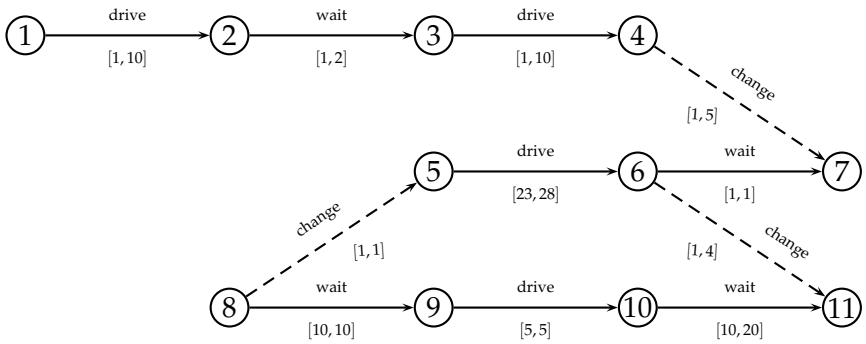
Lemma 2.4.16: [Sch10]
Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitätsnetzwerk und Π ein Fahrplan. Dann gilt für jeden ungerichteten Kreis C in $\mathcal{N}$:

$$tension(C) = 0$$

Beweis:
Sei $C = (i_0, i_1, \ldots, i_n)$ mit $i_0 = i_n$ ein ungerichteter Kreis. Dann gilt:

$$\begin{aligned}
tension(C) &\overset{\text{Def}}{=} \sum_{a \in C^+} \xi_a - \sum_{a \in C^-} \xi_a \\
&= \sum_{(i_j, i_{j+1}) \in C^+} \xi_{(i_j, i_{j+1})} - \sum_{(i_{j+1}, i_j) \in C^-} \xi_{(i_{j+1}, i_j)} \\
&= \sum_{(i_j, i_{j+1}) \in C^+} (\Pi_{i_{j+1}} - \Pi_{i_j}) - \sum_{(i_{j+1}, i_j) \in C^-} (\Pi_{i_j} - \Pi_{i_{j+1}}) \\
&= \sum_{j=0}^{n-1} (\Pi_{i_{j+1}} - \Pi_{i_j}) \\
&= \Pi_{i_n} - \Pi_{i_0} \\
&= 0
\end{aligned}$$

Beispiel 6:



mit zulässigem Fahrplan

$$\begin{array}{llll} \Pi_1 & = 1 & \Pi_2 & = 10 \\ \Pi_3 & = 11 & \Pi_4 & = 20 \\ \Pi_5 & = 1 & \Pi_6 & = 24 \\ \Pi_7 & = 25 & \Pi_8 & = 0 \\ \Pi_9 & = 10 & \Pi_{10} & = 15 \\ \Pi_{11} & = 28 & & \end{array}$$

Dann gilt für den ungerichteten Kreis $C = (8, 5, 6, 11, 10, 9, 8)$:

$$\begin{aligned} tension(C) = & \quad (\Pi_5 - \Pi_8) + (\Pi_6 - \Pi_5) + (\Pi_{11} - \Pi_6) \\ & -(\Pi_{11} - \Pi_{10}) - (\Pi_{10} - \Pi_9) - (\Pi_9 - \Pi_8) \\ = & \quad 1 + 23 + 4 - 13 - 5 - 10 \\ = & \quad 0 \end{aligned}$$

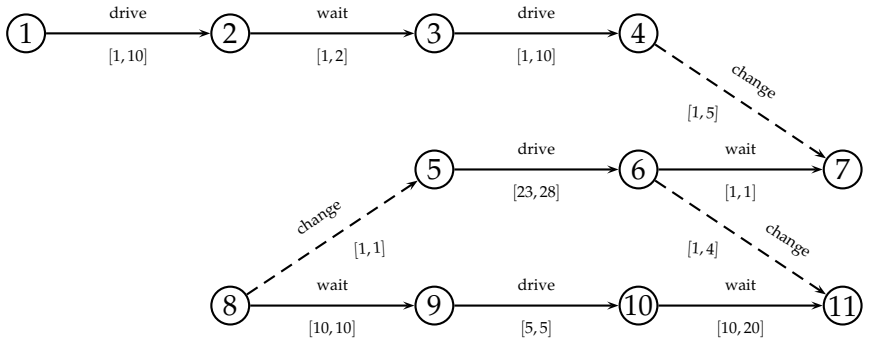
Um das Problem des Findens eines zulässigen Fahrplans zu lösen, erstellen wir aus dem Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ ein neues Netzwerk $\mathcal{N}^*$, in dem wir das kürzeste Wege Problem lösen können.

Notation 2.4.17: [Sch10]
Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitätsnetzwerk. Das neue Netzwerk $\mathcal{N}^* = (\mathcal{E}^*, \mathcal{A}^*)$ wird konstruiert durch

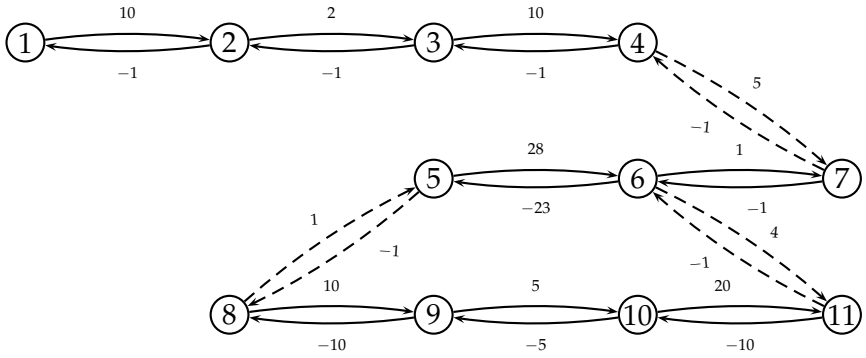
- $\mathcal{E}^* = \mathcal{E}$
- $\mathcal{A}^* = \mathcal{A} \cup \{(i,j) : (j,i) \in \mathcal{A}\}$
- $\mu_a = \begin{cases} U_a & \text{wenn } a \in \mathcal{A} \\ -L_{a'} & \text{wenn } a = (i,j) \notin \mathcal{A} \text{ und } a' = (j,i) \in \mathcal{A} \text{ die inverse Kante zu } a \text{ ist} \end{cases}$

Notation 2.4.18: [Sch10]
Sei $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ ein Ereignis-Aktivitätsnetzwerk und $\mathcal{N}^*$ wie in Notation 2.4.17. Dann bezeichne $SP(u, v)$ die Länge des kürzesten Pfades von u nach v in $\mathcal{N}^*$.

Beispiel 7:
Sei $\mathcal{N}$ gegeben durch folgendes Netzwerk



Dann ist $\mathcal{N}^*$ gegeben durch:



Ein Kriterium zum Erkennen, ob ein Fahrplan zulässig ist und ob es überhaupt einen zulässigen Fahrplan in einem Netzwerk gibt, liefert der folgende Satz:

Satz 2.4.19: [Sch10]

1. Π ist ein zulässiger Fahrplan in $\mathcal{N} \Leftrightarrow \Pi_j - \Pi_i \leq \mu_a \; \forall a = (i,j) \in \mathcal{A}'$
2. Ein zulässiger Fahrplan Π in $\mathcal{N}$ existiert $\Leftrightarrow$ $\mathcal{N}^*$ enthält keinen Kreis negativer Länge.

Beweis:

1. Sei Π ein zulässiger Fahrplan.

$$\begin{aligned}
&\Leftrightarrow \; \forall a = (i,j) \in \mathcal{A}: && \Pi_j - \Pi_i \leq U_a \\
&&& \Pi_j - \Pi_i \geq L_a \\
&\Leftrightarrow \; \forall a = (i,j) \in \mathcal{A}: && \Pi_j - \Pi_i \leq U_a \\
&&& \Pi_i - \Pi_j \leq -L_a \\
&\Leftrightarrow \; \forall a = (i,j) \in \mathcal{A}': && \Pi_j - \Pi_i \leq \mu_a
\end{aligned}$$

2. "$\Rightarrow$" Sei $C = (i_0, i_1, \ldots, i_n)$ mit $i_0 = i_n$ ein Kreis in $\mathcal{N}'$. Zu zeigen ist, dass C von nicht-negativer-Länge ist. C entspricht einem ungerichteten Kreis in $\mathcal{N}$. Aus 2.4.16 wissen wir, dass $tension(C) = 0$ ist und daraus folgt:

$$0 \overset{2.4.16}{=} tension(C) \overset{\text{Def}}{=} \sum_{j=0}^{n-1} \Pi_{i_{j+1}} - \Pi_{i_j} \overset{\text{Teil 1.}}{\leq} \sum_{a \in C} \mu_a$$

"$\Leftarrow$" In $\mathcal{N}^*$ existiere kein Kreis negativer Länge $\Leftrightarrow \sum_{a \in C} \mu_a \geq 0$ für alle Kreise C in $\mathcal{N}^*$. Wir konstruieren einen Fahrplan Π wie folgt:
Wähle einen zufälligen Knoten $i^* \in \mathcal{E}$ und definiere $\Pi_i := SP(i^*, i)$, wobei $SP(i^*, i)$ wie in Notation 2.4.18 die Länge des kürzesten Pfades von i^* nach i

in $\mathcal{N}^*$ bezeichnet. Wir müssen nun zeigen, dass Π tatsächlich ein zulässiger Fahrplan ist. Nach der Dreiecksungleichung gilt

$$SP(i^*, j) \leq SP(i^*, i) + SP(i, j)$$

Wir ersetzen nun $SP(i, j)$ durch μ_{ij} und erhalten:

$$SP(i^*, j) - SP(i^*, i) \leq \mu_{ij}$$

Daraus folgt, dass $\Pi_j - \Pi_i \leq \mu_a$ für alle $a = (i, j) \in \mathcal{A}^*$. Nach Teil 1 dieses Satzes ist Π damit ein zulässiger Fahrplan. $\square$

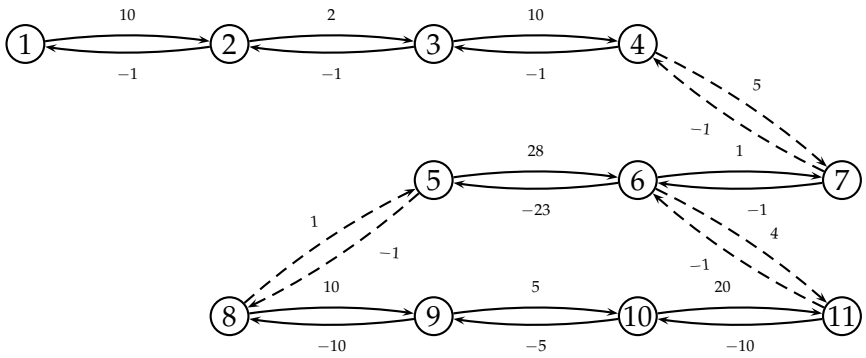
Beispiel 8:

1. Da im Beispiel 7 in $\mathcal{N}^*$ kein Kreis negativer Länge existiert, besitzt das Ereignis-Aktivitätsnetzwerk einen zulässigen Fahrplan. Diesen haben wir bereits in Beispiel 6 kennengelernt.
2. Ändern wir nun im Netzwerk $\mathcal{N}$ die untere Schranke der Kante $(10, 11)$ auf 20, so besitzt das Netzwerk $\mathcal{N}^*$ den negativen Kreis $C = (8, 5, 6, 11, 10, 9, 8)$. Somit existiert in diesem Netzwerk kein zulässiger Fahrplan.

Um einen zulässigen Fahrplan zu finden, ist der folgende Algorithmus nützlich.

Algorithmus 2.4.20: [Sch10]
Input: Ereignis-Aktivitätsnetzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Zeitspannen $\Delta_a = [L_a, U_a]$ für alle $a \in \mathcal{A}$
Output: Ein zulässiger Fahrplan Π für alle $i \in \mathcal{E}$, wenn einer existiert
1: Konstruiere $\mathcal{N}^*$
2: Wähle einen zufälligen Knoten $i^* \in \mathcal{E}$
3: Berechne $SP(i^*, i)$ für alle $i \in \mathcal{E}$.
4: **if** $SP(i^*, i) = -\infty$ **then**
5: Stop. Es gibt keinen zulässigen Fahrplan.
6: **end if**
7: Gebe $\Pi := SP(i^*, i)$ für alle $i \in \mathcal{E}$ aus.

Beispiel 9:
Wir betrachten erneut Beispiel 7:



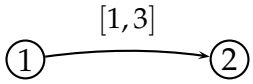
Wir wenden nun Algorithmus 2.4.20 an. Wir wählen $i^* = 1$ und erhalten

$$\begin{aligned} SP(1,1) &= 0 \\ SP(1,2) &= 10 \\ SP(1,3) &= 12 \\ SP(1,4) &= 22 \\ SP(1,5) &= 1 \\ SP(1,6) &= 26 \\ SP(1,7) &= 27 \\ SP(1,8) &= 0 \\ SP(1,9) &= 10 \\ SP(1,10) &= 15 \\ SP(1,11) &= 25 \end{aligned}$$

Die Probe zeigt, dass dieser Fahrplan zulässig ist.

Lemma 2.4.21:
Nicht für jeden zulässigen Fahrplan gibt es einen Startknoten, so dass Algorithmus 2.4.20 den Fahrplan findet.

Beweis:
Wir betrachten das folgende Beispiel:



Algorithmus 2.4.20 bestimmt die Fahrpläne

$$\Pi_1 = 0 \qquad \Pi_2 = 3$$

und

$$\Pi_1 = 0 \qquad \Pi_2 = 1 \ .$$

Der Fahrplan

$$\Pi_1 = 0 \qquad \Pi_2 = 2$$

ist aber ein weiterer zulässiger Fahrplan, der vom Algorithmus nicht gefunden wird.

□

Wir können nun zulässige Fahrpläne bestimmen. Doch welcher dieser Fahrpläne ist "der Beste"? Zunächst müssen wir dafür definieren, wann ein Fahrplan "gut" ist.

Da Fahrgäste so schnell wie möglich ans Ziel kommen möchten, scheint ein Fahrplan gut zu sein, wenn der Slack $s_a = (\Pi_j - \Pi_i) - L_a$ für alle $a \in \mathcal{A}$, das heißt die Reisezeit auf einer Kante a minus die minimal nötige Zeit, minimiert wird. Diese Zielfunktion vermeidet unnötige Wartezeiten, ist aber nicht robust gegen Verspätungen. Die Robustheit kann verbessert werden durch ein "künstliches" Anheben der Werte von L_a, so dass ein bestimmter Slack von vornherein mit gegeben ist. Wenn wir zum Beispiel sehen, dass wir im Winter wegen Schneeverwehungen oder vereisten Weichen mehr Zeit brauchen, setzen wir die unteren Schranken hoch, selbst, wenn es dann bei guten Wetterbedingungen technisch möglich wäre, eine kürzere Zeit einzuplanen. Da die L_a konstant sind, gilt

$$\min \sum_{a=(i,j)\in\mathcal{A}} (\Pi_j - \Pi_i) - L_a \quad \Leftrightarrow \quad \min \sum_{a=(i,j)\in\mathcal{A}} (\Pi_j - \Pi_i).$$

Diese Zielfunktion sagt aus, dass es für die Fahrgäste gleich gut oder schlecht ist, eine Fahraktivität, eine Warteaktivität oder eine Umsteigeaktivität zu "nutzen". Wenn wir annehmen, dass die Fahrgäste es schlimmer finden, am Bahnhof beim Umsteigen zu warten, kann man auch die Zielfunktion $\sum_{a=(i,j)\in\mathcal{A}_{change}} (\Pi_j - \Pi_i)$ minimieren oder die Umsteigeaktivitäten noch zusätzlich gewichten. Analog für Fahraktivitäten oder Warteaktivitäten. Wir werden in dieser Arbeit Warteaktivitäten, Fahraktivitäten und Umsteigeaktivitäten als gleich wichtig betrachten.

Wir wollen unsere Zielfunktion so gestalten, dass die Anzahl der Passagiere, die eine Aktivität $a \in \mathcal{A}$ ausführen, mit ins Spiel kommt. Das heißt, je mehr Passagiere eine Aktivität $a \in \mathcal{A}$ machen, desto wichtiger ist es, die Zeit auf dieser Kante klein zu halten. Wir modellieren die Anzahl der Passagiere auf Aktivität a durch Gewichte w_a und haben die folgende Zielfunktion:

$$\sum_{a=(i,j)\in\mathcal{A}} w_a(\Pi_j - \Pi_i) \ .$$

Vgl. auch [Sch10]

Zusammenfassend ergibt sich nun folgendes Problem:

Problem 2.4.22: [Sch10]
Gegeben sei ein Ereignis-Aktivitätsnetzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ mit Zeitspannen Δ_a und Anzahl der Fahrgäste w_a für alle $a \in \mathcal{A}$. Zu finden ist ein zulässiger Fahrplan Π, der $\sum_{a=(i,j)\in\mathcal{A}} w_a(\Pi_j - \Pi_i)$ minimiert.

Wir können das Optimierungsproblem als ein ganzzahliges lineares Programm formulieren

$$
\begin{aligned}
(P1) \quad \min \quad & \sum_{a=(i,j)\in\mathcal{A}} w_a(\Pi_j - \Pi_i) & \\
\text{s.d.} \quad & \Pi_j - \Pi_i \leq U_a & \forall a = (i,j) \in \mathcal{A} \\
& \Pi_j - \Pi_i \leq -L_a & \forall a = (i,j) \in \mathcal{A} \\
& \Pi_i \in \mathbb{Z} & \forall i \in \mathcal{E}
\end{aligned}
$$

Lemma 2.4.23: [Sch10]
Die LP-Relaxation von $(P1)$ gibt eine ganzzahlige Lösung.

Beweis:
Sei A die Koeffizientenmatrix von $(P1)$. Jede Zeile enthält genau eine 1 und eine -1. Also ist die Transponierte A^T der Koeffizientenmatrix A total unimodular und damit auch A selber. Aus Lemma 2.3.15 folgt die Ganzzahligkeit der LP-Relaxation. □

3 Aperiodische Fahrplangestaltung mit OD-Paaren

3.1 Motivation

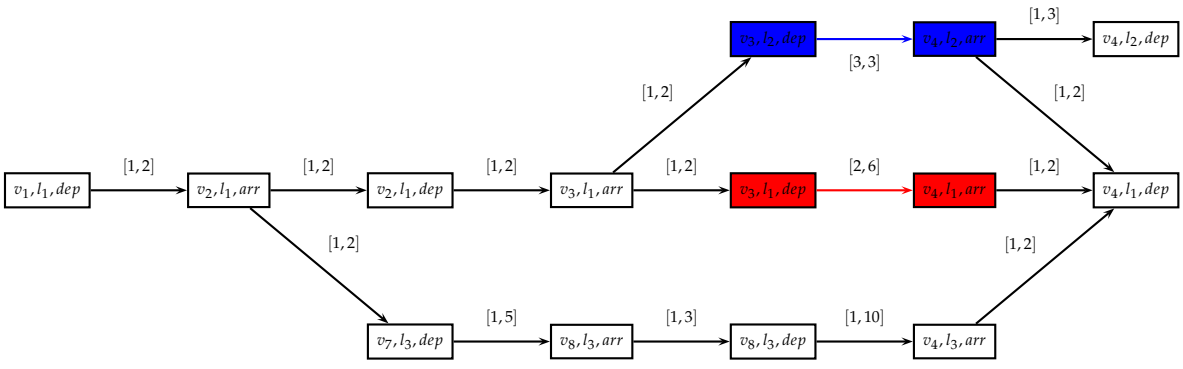
Wir haben in Lemma 2.4.23 gesehen: Wenn wir schon vorher wissen, wieviele Passagiere Aktivität $a = (i, j)$ $\forall a \in \mathcal{A}$ ausführen, können wir das aperiodische Fahrplanproblem leicht lösen. In der Praxis sind diese Daten jedoch nicht vorher bekannt.

Wir beziehen nun die Tatsache mit ein, dass sich die Passagiere in der Wahl ihrer Wege (insbesondere des Abfahrts- und Ankunftsereignisses) nach dem Fahrplan richten und der Fahrplan wiederum (wie man in der Zielfunktion $\sum_{a=(i,j)\in\mathcal{A}} w_a(\Pi_j - \Pi_i)$ sieht) von den Wegen, die die Passagiere nehmen, abhängt.

Bemerkung 3.1.1: [Sch10]
Da es nicht für jeden zulässigen Fahrplan einen Startknoten gibt, so dass Algorithmus 2.4.20 den Fahrplan findet, ist es nicht ratsam diesen Algorithmus zum Optimieren zu verwenden.

Wir brauchen also eine andere Methode. Wir gehen im Folgenden davon aus, dass wir die Anzahl der Passagiere, die von u nach v wollen, kennen, nicht jedoch den Weg, den diese nehmen.

Beispiel 10:
Wie betrachten das folgende Ereignis-Aktivitätsnetzwerk:



Um von v_3 nach v_4 zu gelangen, haben die Passagiere die Wahl zwischen dem Weg $P^1_{(v_3,v_4)} = ((v_3, l_2, dep), (v_4, l_2, arr))$ mit einer Fahrzeit von 3 Zeiteinheiten oder als Al-

ternative den Weg $P^2_{(v_3,v_4)} = ((v_3, l_1, dep), (v_4, l_1, arr))$, der im besten Fall 2 Zeiteinheiten, aber im schlechtesten Fall 6 Einheiten braucht. Ob der beste Fall (Best case), der schlechteste Fall (Worst case) oder etwas dazwischen eintritt, wird auch noch von der Linie l_3 beeinflusst. Ob Weg $P^1_{(v_3,v_4)}$ oder $P^2_{(v_3,v_4)}$ günstiger ist, hängt somit vom Fahrplan ab. Dieser ist aber wiederum davon abhängig, welchen Weg die Passagiere nehmen, denn wenn sie Weg $P^2_{(v_3,v_4)}$ nehmen würden, würde der Fahrplan so gewählt werden, dass $P^2_{(v_3,v_4)}$ eine möglichst kleine Länge hat. Nehmen sie jedoch Weg $P^1_{(v_3,v_4)}$, dann ist schon von vornherein vorgegeben, dass dieser Weg eine Länge von 3 hat (durch den Span $[3,3]$), und die Länge von $P^2_{(v_3,v_4)}$ wäre somit egal und könnte "lang" gesetzt werden.

3.2 Grundlagen

Die Passagierdaten werden durch die sogenannte Origin-Destination-Menge (OD-Menge) OD gegeben, welche OD-Paare (u, v) enthält, wobei u der Ursprung und v das Ziel ist. Für jedes OD-Paar ist die Anzahl der Passagiere w_{uv}, die von u nach v reisen, gegeben.

Definition 3.2.1: [SS10]

- (u, v) mit u als Ursprung und v als Ziel (im öffentlichen Verkehrsnetzwerk) nennt man OD-Paar.
- Die Menge von OD-Paaren $OD = \{(u, v, w_{(u,v)})\}$ heißt OD-Menge.
- Mit w_{uv} wird die Anzahl der Passagiere, die von Haltestelle u nach v fahren, bezeichnet.

Bemerkung 3.2.2:
Wie werden das OD-Paar $(u, v) \in OD$ mit Passagieranzahl/Gewicht w_{uv} auch als (u, v, w_{uv}) oder $(u, v) = w_{uv}$ schreiben.

Wir integrieren nun das Routing, das heißt das Finden der Wege für die Passagiere im Ereignis-Aktivitätsnetzwerk, in den Prozess der Fahrplangestaltung.

Jeder Weg von einem Abfahrtsereignis i an Bahnhof u zu einem Ankunftsereignis j an Bahnhof v ist eine mögliche Reise von i nach j. Da im Allgemeinen $\Pi_i \neq \Pi_{i'}$ und $\Pi_j \neq \Pi_{j'}$ für Abfahrtsereignisse i und i' an Bahnhof u und Ankunftsereignisse j und j' an Bahnhof v (das heißt, zwei Züge fahren nicht zur selben Zeit ab, beziehungsweise kommen nicht zur selben Zeit an.), hängt die Reisezeit vom Weg ab, der für dieses OD-Paar gewählt wurde. Genauer: Wie wir später sehen werden, hängt die Reisezeit vom Abfahrts- und vom Ankunftsereignis ab. Dabei gehen wir davon aus, dass die Reisezeit für die Passagiere bei einem Abfahrtsereignis anfängt und bei einem Ankunftsereignis aufhört.

Um diesen Sachverhalt ins Netzwerk zu integrieren, führen wir Origin-Knoten (Ursprungsknoten) u und Destination-Knoten (Zielknoten) v für jedes OD-Paar $(u, v) \in OD$ ein.

Definition 3.2.3: [SS10]

- Für alle Abfahrtsereignisse an einem Bahnhof v_i wird ein Origin-Knoten u eingeführt.
- Für alle Ankunftsereignisse an einem Bahnhof v_j wird ein Destination-Knoten v eingeführt.

Definition 3.2.4: [SS10]

- $\mathcal{E}_{org}$ ist die Menge der Origin-Knoten,
- $\mathcal{E}_{dest}$ ist die Menge der Destination-Knoten

$\mathcal{E}' = \mathcal{E}_{org} \cup \mathcal{E}_{dest}$ wird als Menge der Origin- und Destination-Knoten bezeichnet.

Wir verbinden jedes $u \in \mathcal{E}_{org}$ mit allen Abfahrtsereignissen i an Bahnhof u durch eine Origin-Kante (u, i) und jedes $v \in \mathcal{E}_{dest}$ mit allen Ankunftsereignissen j an Bahnhof v durch eine Destination-Kante (j, v).

Definition 3.2.5: [SS10]

- Die Menge der Origin-Kanten wird mit $\mathcal{A}_{org}$ bezeichnet.
  Formal: $\mathcal{A}_{org} = \{(u, i) \mid u \in \mathcal{E}_{org} \text{ und } i = (s_u, l, dep)\}$
- Die Menge der Destination-Kanten wird mit $\mathcal{A}_{dest}$ bezeichnet.
  Formal: $\mathcal{A}_{dest} = \{(j, v) \mid v \in \mathcal{E}_{dest} \text{ und } j = (s_v, l, arr)\}$

$\mathcal{A}' = \mathcal{A}_{org} \cup \mathcal{A}_{dest}$ definieren wir als die Menge der Origin- und Destination-Kanten.

Notation 3.2.6:
Das Ereignis-Aktivitätsnetzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ zusammen mit den Origin- und Destination-Knoten $\mathcal{E}'$ und -Kanten $\mathcal{A}'$ wird im Folgenden als $\mathcal{N}'$ bezeichnet.

Bemerkung 3.2.7:
Es reicht aus, Origin-/Destination-Kanten und Knoten nur für die Bahnhöfe einzuführen, die auch als Bahnhöfe in der OD-Menge vorhanden sind.

Das aperiodische Fahrplanproblem mit OD-Paaren lässt sich nun folgendermaßen formulieren:

Problem 3.2.8: [SS10]
Wir wollen in $\mathcal{N}'$ einen zulässigen Fahrplan Π und für jedes OD-Paar (u, v) einen u-v-Pfad $P_{uv} = (u, i_1^{uv}, i_2^{uv}, \ldots, i_{uv}^{uv}, v)$ finden, so dass

$$\sum_{(u,v) \in OD} w_{uv}(\Pi_{i_{uv}^{uv}} - \Pi_{i_1^{uv}}) \tag{7}$$

minimal wird.

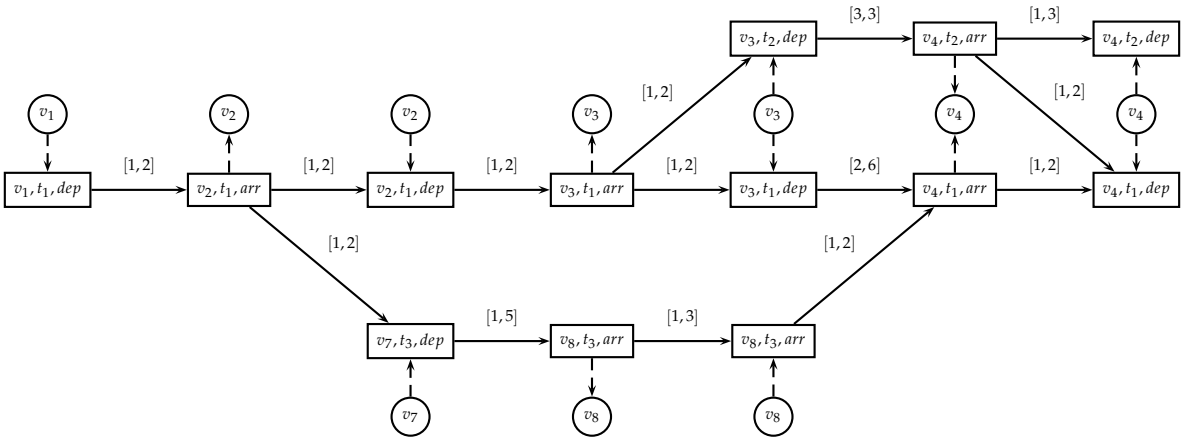


Abbildung 3.1: Beispiel 10 mit Origin- und Destination-Knoten und -Kanten

Bemerkung 3.2.9:
Da es für eine Instanz sehr viele zulässige Fahrpläne geben kann, die sich in den einzelnen Zusammenhangskomponenten in jedem Ereignis nur um ein Vielfaches unterscheiden, ist es sinnvoll "Äquivalenzklassen" von Fahrplänen einzuführen.

Definition 3.2.10:
Seien Π^1 und Π^2 zwei Fahrpläne für das Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ und seien $q_1, \ldots, q_n$ die Zusammenhangskomponenten in $\mathcal{N}$. Seien weiterhin i_{q_l} die Ereignisse, die in Zusammenhangskomponente q_l liegen.
Π^1 und Π^2 sind äquivalent ($\Pi^1 \sim \Pi^2$) , wenn es für jede Zusammenhangskomponente q_l für $l = 1, \ldots, n$ ein $k_l \in \mathbb{Z}$ gibt, so dass $\Pi^1_{q_l} = \Pi^2_{q_l} + k_l$.

Wir vergewissern uns, dass die so eingeführte Relation eine Äquivalenzrelation ist:

Lemma 3.2.11:
$\sim$ ist eine Äquivalenzrelation.

Beweis:
Es gilt:

1. $\Pi \sim \Pi$: Mit $k_l = 0 \ \forall$ Zusammenhangskomponenten $l = 1, \ldots, n$ gilt $\Pi = \Pi + k_l = \Pi + 0 = \Pi$.

2. $\Pi^1 \sim \Pi^2 \Rightarrow \Pi^2 \sim \Pi^1$: Es gelte $\Pi^1 \sim \Pi^2$. Das heißt, für jede Zusammenhangskomponente $l = 1, \ldots, n$ gibt es ein $k_l \in \mathbb{Z}$, so dass $\Pi^1 = \Pi^2 + k_l$. Wegen $\Pi^2 = \Pi^1 + (-k_l)$ gilt auch $\Pi^2 \sim \Pi^1$.

3. $\Pi^1 \sim \Pi^2, \Pi^2 \sim \Pi^3 \Rightarrow \Pi^1 \sim \Pi^3$: Es gelten $\Pi^1 \sim \Pi^2, \Pi^2 \sim \Pi^3$. Es gibt also für alle Zusammenhangskomponenten $l = 1, \ldots, n$ Zahlen k_l und $h_l \in \mathbb{Z}$, so dass $\Pi^1 = \Pi^2 + k_l$ und $\Pi^2 = \Pi^3 + h_l$. Zusammen ergibt sich $\Pi^1 = \Pi^3 + h_l + k_l = \Pi^3 + j_l$ mit $j_l = h_l + k_l$.

□

Beispiel 11:
Seien Π^1 und Π^2 gegeben durch:

$$\begin{aligned} \Pi_1^1 = 0 \quad \Pi_1^2 &= 5 \\ \Pi_2^1 = 5 \quad \Pi_2^2 &= 10 \\ \Pi_3^1 = 0 \quad \Pi_3^2 &= 4 \\ \Pi_4^1 = 3 \quad \Pi_4^2 &= 7 \end{aligned}$$

und die Zusammenhangskomponenten seien gegeben durch $\{1, 2\}$ und $\{3, 4\}$. Dann sind Π^1 und Π^2 äquivalent, da $\Pi_i^1 + 5 = \Pi_i^2$ für $i = 1, 2$ und $\Pi_i^1 + 4 = \Pi_i^2$ für $i = 3, 4$.

Ein Kriterium zum Herauszufinden, ob es im Netzwerk $\mathcal{N}$ überhaupt einen zulässigen Fahrplan gibt, liefert analog zu Teil 2 von Satz 2.4.19 der folgende Satz.

Satz 3.2.12:
Für eine Instanz des Fahrplanproblems mit OD-Paaren existiert ein zulässiger Fahrplan $\Longleftrightarrow$

1. Es gibt keinen Kreis negativer Länge im Netzwerk $\mathcal{N}^* = (\mathcal{E}^*, \mathcal{A}^*)$ mit $\mathcal{E}^* = \mathcal{E}$ und $\mathcal{A}^* = \mathcal{A} \cup \{(j, i) \mid (i, j) \in \mathcal{A}\}$ mit Kantenlängen $c_a = U_a \ \forall a \in \mathcal{A}$ und $c_a = -L_a \ \forall a \in \mathcal{A}^* \setminus \mathcal{A}$

2. Für jedes OD-Paar $(u, v) \in OD$ existiert ein gerichteter Weg von u nach v in $\mathcal{N}'$.

Beweis:
Sei I mit Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ und OD-Menge OD eine Instanz des aperiodischen Fahrplanproblems mit OD-Paaren in $\mathcal{N}$. Es gilt:

$$\exists \text{ ein zulässiger Fahrplan in } \mathcal{N}'. \Leftrightarrow \exists \text{ zulässiger Fahrplan in } \mathcal{N}. \tag{3.1}$$
$$\overset{2.4.19}{\Longleftrightarrow} \mathcal{N}^* \text{ enthält keinen Kreis negativer Länge.}$$

Da das aperiodische Fahrplanproblem mit OD-Paaren für Instanz I für jedes OD-Paar $(u, v) \in OD$ einen u-v-Pfad sucht, so dass die Reisezeit minimiert wird, muss es einen Weg von u nach v geben, damit das Problem für die Instanz I gelöst werden kann.

□

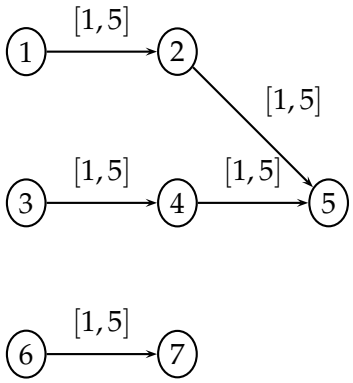
Es stellt sich heraus, dass das Problem der aperiodischen Fahrplangestaltung mit OD-Paaren NP-schwer ist.

Satz 3.2.13: [SS10]
Das Problem der aperiodischen Fahrplangestaltung mit OD-Paaren ist NP-schwer, selbst wenn alle OD-Paare denselben Ursprung oder dasselbe Ziel haben.

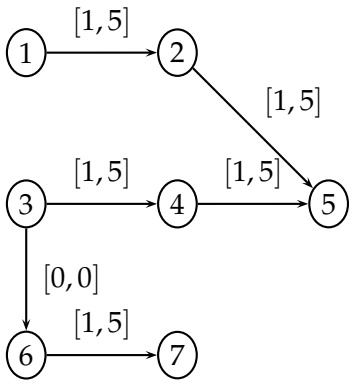
3.2.1 Kanten-basierende ILP-Formulierung

Bemerkung 3.2.14:
Da es viele verschiedene äquivalente Fahrpläne gibt, ist es sinnvoll einen Repräsentanten für diese zu wählen. Wir wählen diesen folgendermaßen: Seien $Z_1, Z_2, \ldots, Z_n$ die n Zusammenhangskomponenten des Ereignis-Aktivitätsnetzwerkes $\mathcal{N}$. Wir fügen zwischen den einzelnen Zusammenhangskomponenten Z_{i-1} und Z_i $\forall i = 2, \ldots, N$ jeweils eine Kante mit Span $[0, 0]$ (also Länge 0) ein. Auf diese Weise können wir den Fall, dass es mehrere Zusammenhangskomponenten in $\mathcal{N}$ gibt, auf den Fall einer Zusammenhangskomponente zurückführen.

Beispiel 12:
Wir betrachten folgendes Ereignis-Aktivitätsnetzwerk $\mathcal{N}$:



Wir haben die zwei Zusammenhangskomponenten gegeben durch die Knotenmengen $V_1 = \{1, 2, 3, 4, 5\}$ und $V_2 = \{6, 7\}$ und den Kantenmengen E_1 und E_2. Wir fügen eine Kante $(3, 6)$ mit Span $[0, 0]$ ein.



Definition 3.2.15:
$\mathcal{A}_{con}$ sei die Menge aller Kanten, die zwei Zusammenhangskomponenten aus $\mathcal{N}'$ verbinden.

Beispiel 13:
Wir betrachten das Ereignis-Aktivitätsnetzwerk aus Beispiel 12. Dann ist $\mathcal{A}_{con} = \{(3,6)\}$.

Notation 3.2.16:
Wir schreiben $\tilde{\mathcal{A}} := \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}_{change} \cup \mathcal{A}_{con}$.

Notation 3.2.17:
Das Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ mit Origin- und Destination-Knoten und -Kanten und den Kanten $\mathcal{A}_{con}$ bezeichnen wir im Folgenden mit $\widetilde{\mathcal{N}'}$.

Bemerkung 3.2.18:
Wenn wir zukünftig von allen zulässigen Fahrplänen sprechen, meinen wir alle Repräsentanten bezüglich der Äquivalenzrelation aus Definition 3.2.10.

Wir werden nun ein ganzzahliges lineares Programm für unser Problem formulieren [Sch11]. Dieses benutzt Kanten-basierende Variablen q_a^{uv}. Die Variablen Π_i ordnen jedem Ereignis $i \in \mathcal{E}_{dep} \cup \mathcal{E}_{arr}$ eine geplante Zeit zu und die Variablen t_u^{uv} und t_v^{uv} repräsentieren für jedes OD-Paar $(u,v) \in OD$ die Abfahrtszeit an Bahnhof u beziehungsweise Ankunftszeit an Bahnhof v. Also beschreibt die Differenz $t_v^{uv} - t_u^{uv}$ die Reisezeit der Passagiere "eines OD-Paares". Weiterhin sei

$$q_a^{uv} = \begin{cases} 1 & \text{wenn Kante } a \text{ vom OD-Paar } (u,v) \text{ benutzt wird} \\ 0 & \text{sonst} \end{cases}$$

Wir erhalten folgendes ganzzahlige lineare Program:

$$
\begin{aligned}
(ILP1) \quad \min \quad & \sum_{(u,v)\in OD} w_{uv}(t_v^{uv} - t_u^{uv}) && && (8)\\
\text{s. d.} \quad & \sum_{a\in\delta^{out}(u)} q_a^{uv} = 1 && \forall (u,v) \in OD && (9)\\
& \sum_{a\in\delta^{out}(e)} q_a^{uv} - \sum_{a\in\delta^{in}(e)} q_a^{uv} = 0 && \forall (u,v) \in OD && (10)\\
& \sum_{a\in\delta^{in}(v)} q_a^{uv} = 1 && \forall (u,v) \in OD && (11)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (12)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (13)\\
& t_u^{uv} \leq \Pi_i + M(1 - q_{(u,i)}^{uv}) && \forall (u,i) \in \mathcal{A}_{org} && (14)\\
& t_v^{uv} \geq \Pi_j - M(1 - q_{(j,v)}^{uv}) && \forall (j,v) \in \mathcal{A}_{dest} && (15)\\
& \Pi_i \in \mathbb{Z} && \forall i \in \mathcal{E} &&\\
& t_u^{uv}, t_v^{uv} \in \mathbb{Z} && \forall (u,v) \in OD &&\\
& q_a^{uv} \in \{0,1\} && \forall (u,v) \in OD, a \in \mathcal{A} &&
\end{aligned}
$$

Wir minimieren in der Zielfunktion (8) für alle Passagiere die Summe über allen Zeiten zwischen Abfahrt und Ankunft. Da der Zielfunktionswert minimiert wird, wird versucht t_u^{uv} möglichst groß und t_v^{uv} möglichst klein zu halten. Die Bedingungen (9), (10) und (11) sind Flußbedingungen. Dabei ist (10) die Flußerhaltungsgleichung. Diese sagt aus, dass alles, was in einen Knoten hereinfließt, auch wieder herausfließt. (9) respektive (11) erzwingen, dass für jedes OD-Paar genau eine Origin-Kante und eine Destination-Kante benutzt wird.

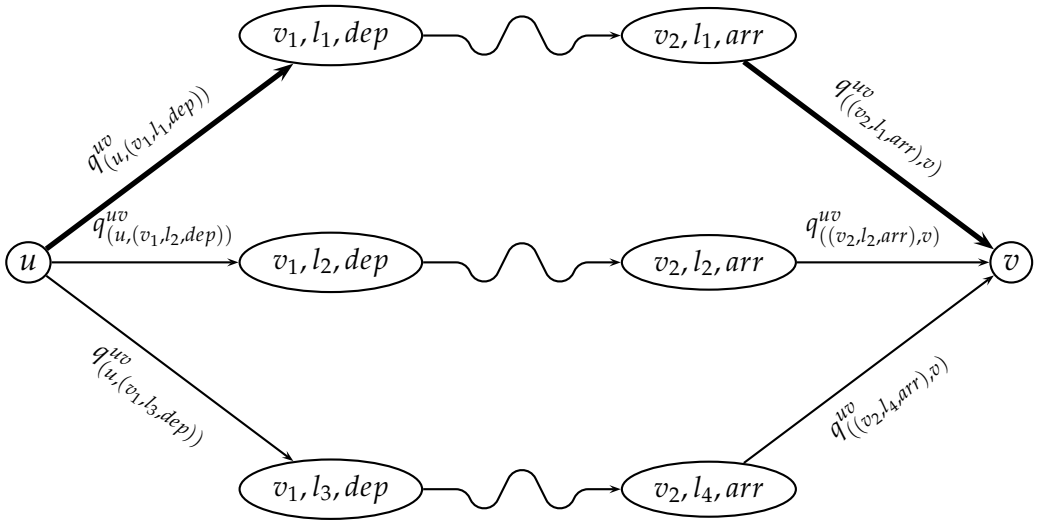


Abbildung 3.2: Genau eine der Origin-Kanten und genau eine der Destination-Kanten wird von OD-Paar (u, v) benutzt. Hier $q^{uv}_{(u,(v_1,l_1,dep))} = 1$ und $q^{uv}_{((v_2,l_1,arr),v)} = 1$.

Um zu zeigen, dass das ganzzahlige lineare Programm das aperiodische Fahrplanproblem mit OD-Paaren löst, ist folgendes zu zeigen:

1. In $\widetilde{\mathcal{N}'}$ wird ein zulässiger Fahrplan gefunden.
2. Für jedes OD-Paar $(u, v) \in OD$ wird in $\mathcal{N}'$ ein u-v-Pfad $P_{uv} = \{u, i_1^{uv}, i_2^{uv}, \ldots, i_{uv}^{uv}, v\}$ gefunden
3. Die so gefundenen Wege und der Fahrplan sind optimal.

Lemma 3.2.19:
Der von $(ILP1)$ gefundene Fahrplan ist zulässig.

Beweis:
Nach Satz 2.4.19 ist ein Fahrplan genau dann zulässig, wenn $\Pi_j - \Pi_i \leq U_a \ \forall a = (i, j) \in \mathcal{A}$ und $\Pi_j - \Pi_i \geq L_a \ \forall a = (i, j) \in \mathcal{A}$. Dies entspricht den Bedingungen (12) und (13) und der Fahrplan ist damit zulässig. □

Lemma 3.2.20:
Für jedes OD-Paar $(u, v) \in OD$ wird ein u-v-Weg in $\mathcal{N}'$ gefunden.

Beweis:
Durch (9) und (11) wird von jedem OD-Paar genau eine Origin-Kante und genau eine Destination-Kante benutzt. Die Flusserhaltungsgleichung (10) leitet die Passagiere durch das Netzwerk und stellt sicher, dass kein Passagier vor seinem Ziel verloren geht. Es wird also für jedes OD-Paar (u, v) ein Weg von u nach v in $\mathcal{N}'$ gewählt.

□

Aus Nebenbedingung (14) und (15) folgt, dass für das erste Abfahrtsereignis i und das letzte Ankunftsereignis j im Weg $t_u^{uv} \leq \Pi_i$ und $t_v^{uv} \geq \Pi_j$ für alle $(u, v) \in OD$ gelten. Dieses gilt unabhängig vom M. Da die Zielfunktion die t_u^{uv} so groß wie möglich macht, gilt für genügend großes M $t_u^{uv} = \Pi_i$ und da die t_v^{uv} so klein wie möglich werden, folgt $t_v^{uv} = \Pi_j$.
Die Wahl der Wege, sowie das richtige Setzen der t_u^{uv} beziehungsweise t_v^{uv} und damit die Übereinstimmung der optimalen Zielfunktionswerte von $(ILP1)$ und des aperiodischen Fahrplanproblems mit OD-Paaren, hängen vom M ab.
Wir definieren zunächst die Mengen $\mathcal{E}(u)$ und $\mathcal{E}(v)$ wie folgt:

Definition 3.2.21: [Sch11]

- $\mathcal{E}(u) := \{i \in \mathcal{E}_{dep} \mid (u, i) \in \mathcal{A}_{org}\}$ ist die Menge der Abfahrtsereignisse für Origin-Knoten u.
- $\mathcal{E}(v) := \{i \in \mathcal{E}_{arr} \mid (i, v) \in \mathcal{A}_{dest}\}$ ist die Menge der Ankunftsereignisse für Destination-Knoten v.

Lemma 3.2.22: [Sch11]
$(ILP1)$ findet für genügend großes M, das heißt für $M \geq \Pi_j - \Pi_i \forall$ zulässigen Fahrpläne Π und alle $(i, j) \in \mathcal{E}(u) \times \mathcal{E}(u) \cup \mathcal{E}(v) \times \mathcal{E}(v)$ eine Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren.

Beweis:
Sei Π die Optimallösung, die $(ILP1)$ ausrechnet und z der entsprechende Zielfunktionswert. Angenommen es gibt eine bessere Lösung als die, die $(ILP1)$ ausrechnet. Diese Lösung habe den Zielfunktionswert z^*. Damit gelte also: $z^* < z$. Dann ist die Lösung gegeben durch einen Fahrplan Π^* und für jedes OD-Paar $(u, v) \in OD$ gibt es einen Weg $P(u, v)^*$. Wir konstruieren nun eine Lösung von $(ILP1)$, so dass diese Lösung

1. zulässig für $(ILP1)$ ist
2. Zielfunktionswert z^* hat.

Da die so konstruierte Lösung somit Zielfunktionswert $z = z^*$ hat, ist dies ein Widerspruch zu $z^* < z$ und die Behauptung wäre damit bewiesen.
Wir konstruieren nun also die Lösung wie folgt. Wir setzen:

1. $\Pi = \Pi^*$
2. $q_a^{uv} = \begin{cases} 1 & \forall \text{ Kanten } a \in P(u,v)^* \\ 0 & \text{sonst} \end{cases}$
3. $t_u^{uv} =$ Abfahrtszeit von OD-Paar (u,v) auf $P(u,v)^* = \Pi_1^{uv}$
4. $t_v^{uv} =$ Ankunftsszeit von OD-Paar (u,v) auf $P(u,v)^* = \Pi_{uv}^{uv}$

Da Π^* zulässig ist, ist auch Π zulässig. Es gelten also $\Pi_j - \Pi_i \leq U_a$ und $\Pi_j - \Pi_i \geq L_a \; \forall a = (i,j) \in \mathcal{A}$. Das heißt, (12) und (13) sind erfüllt.
(9), (10) und (11) sind erfüllt, da für eine Kante a auf dem Weg $P(u,v)^*$ die Variable q_a^{uv} auf 1 gesetzt wird und ein Weg von einem Origin-Knoten u über genau eine Origin-Kante zu einem Abfahrtsereignis führt (Bedingung (9)) und genau über eine Destination-Kante zu einem Destination-Knoten v führt (Bedingung (11)). Da es auf einem Weg keine Abzweigungen gibt, gilt auch die Flußerhaltungsgleichung (10).
Wir zeigen nun, dass die Bedingungen (14) und (15) erfüllt sind. Sei dazu $t_u^{uv} = \Pi_1^{uv}$ die Abfahrtszeit von OD-Paar (u,v) auf Weg $P(u,v)^*$. Dann gilt für $i = 1$:

$$\Pi_1^{uv} + M(1 - q_{(u,1)}^{uv}) \overset{(*)}{=} \Pi_1^{uv} = t_u^{uv} \; ,$$

wobei $(*)$ gilt, da die Kante $(u,1)$ auf dem Weg $P(u,v)^*$ liegt und damit $q_{(u,1)}^{uv} = 1$ gilt. Für $i \neq 1$ liegt (u,i) nicht auf $P(u,v)^*$ und mit $q_{(u,i)}^{uv} = 0$ gilt folglich:

$$\begin{aligned} \Pi_i^{uv} + M(1 - q_{(u,i)}^{uv}) &= \Pi_i^{uv} + M \\ &\geq \Pi_i^{uv} + \Pi_1^{uv} - \Pi_i^{uv} \\ &= \Pi_1^{uv} \\ &= t_u^{uv} \; , \end{aligned}$$

wegen $\Pi_l - \Pi_k \leq M \; \forall (k,l) \in \mathcal{E}(u) \times \mathcal{E}(u) \cup \mathcal{E}(v) \times \mathcal{E}(v)$ und damit insbesondere auch $\Pi_1^{uv} - \Pi_i \leq M$. Damit ist Bedingung (14) erfüllt.
Analog gilt mit $t_v^{uv} = \Pi_{uv}^{uv}$ die Ankunftszeit von OD-Paar (u,v) auf Weg $P(u,v)^*$ für $j = uv$ und $q_{(uv,v)}^{uv} = 1$:

$$\Pi_{uv}^{uv} - M(1 - q_{(uv,v)}^{uv}) = \Pi_{uv}^{uv} = t_v^{uv} \; .$$

Für $j \neq uv$ liegt (j,v) nicht auf dem Weg $P(u,v)^*$ und mit $q_{j,v}^{uv} = 0$ gilt:

$$\begin{aligned} \Pi_j^{uv} - M(1 - q_{j,v}^{uv}) &= \Pi_j^{uv} - M \\ &\leq \Pi_j^{uv} + \Pi_{uv}^{uv} - \Pi_j^{uv} \\ &= \Pi_{uv}^{uv} \\ &= t_v^{uv} \; , \end{aligned}$$

wegen $\Pi_{uv}^{uv} - \Pi_j^{uv} \leq M$. Auch Bedingung (15) ist somit erfüllt.

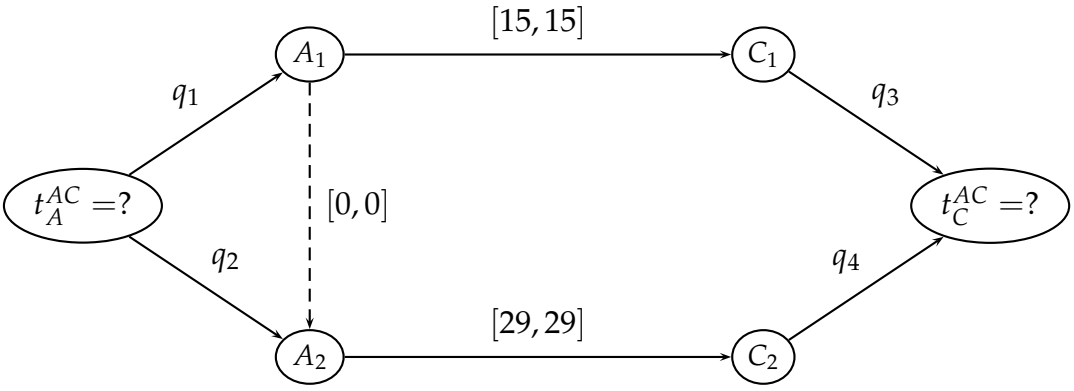
Es bleibt zu zeigen, dass der Zielfunktionswert von $(ILP1)$ mit der so konstruierten Lösung z^* beträgt und somit ein Widerspruch auftritt.
Sei dazu $l_{P(u,v)^*}^{\Pi^*}$ die Länge vom $u-v$-Weg $P(u,v)^*$ bezüglich Fahrplan Π^*. Dann gilt:

$$\begin{aligned} z &< z^* \\ &= \sum_{(u,v)\in OD} w_{uv} \cdot l_{P(u,v)^*}^{\Pi^*} \\ &= \sum_{(u,v)\in OD} w_{uv} \cdot (\Pi_{uv}^{*\,uv} - \Pi_1^{*\,uv}) \\ &= \sum_{(u,v)\in OD} w_{uv} \cdot (\Pi_{uv}^{uv} - \Pi_1^{uv}) \\ &= \sum_{(u,v)\in OD} w_{uv} \cdot (t_v^{uv} - t_u^{uv}) \\ &= z \quad \lightning \end{aligned}$$

□

Ein Beispiel dafür was passiert, wenn M zu klein ist:

Beispiel 14:
Wir betrachten



mit OD-Paar $(A, C, 1)$.
Der optimale Zielfunktionswert ist $z^* = 15$.
Für $M = 13$ und $M = 14$ ergeben sich folgende Fahrpläne Π und Zielfunktionswerte $z = t_v - t_u$:

	$M = 13$	$M = 14$
Π_{A_1}	0	0
Π_{A_2}	0	0
Π_{C_1}	15	15
Π_{C_2}	29	29
t_u	0	0
t_v	16	15
z	16	15

Für $M = 13$ erhalten wir also einen falschen "optimalen" Zielfunktionswert, nämlich $z = 16$. Dieser ergibt sich folgendermaßen:
Die Bedingungen (14) und (15) sind:

$$\begin{aligned} t_u &\leq \Pi^k_{A_1} + M(1 - q_1) &&= 0 + 13 \cdot (1 - q_1) \\ t_u &\leq \Pi^k_{A_2} + M(1 - q_2) &&= 0 + 13 \cdot (1 - q_2) \\ t_v &\geq \Pi^k_{C_1} - M(1 - q_3) &&= 15 - 13 \cdot (1 - q_3) \\ t_v &\geq \Pi^k_{C_2} - M(1 - q_4) &&= 29 - 13 \cdot (1 - q_4) \end{aligned}$$

Die beiden möglichen Wege sind $P_1 = (A, A_1, C_1, C)$ und $P_2 = (A, A_2, C_2, C)$, also im ersten Fall für $q_1 = q_3 = 1$ und im zweiten Fall für $q_2 = q_4 = 1$.
Es gilt dann:

P_1	$q_1 = 1$ $q_2 = 0$ $q_3 = 1$ $q_4 = 0$	$t_u \leq 0$ $t_u \leq 13$ $t_v \geq 15$ $t_v \geq 16$	$t_v^{uv} - t_u^{uv}$ $=16-0$ $=16$
P_2	$q_1 = 0$ $q_2 = 1$ $q_3 = 0$ $q_4 = 1$	$t_u \leq 13$ $t_u \leq 0$ $t_v \geq 2$ $t_v \geq 29$	$t_v^{uv} - t_u^{uv}$ $=29-0$ $=29$

Welches M ist für $(ILP1)$ geeignet?

Es stellt sich nun die Frage, welches M geeignet ist. Einerseits muss M so groß sein, dass die Bedingungen erfüllt sind, andererseits kann man durch kleineres M die Zeit, die man zum Lösen braucht, beschleunigen, da die Optimalitätslücke mit kleinerem M kleiner sein kann.

Wählt man ein zu kleines M, so können t_u^{uv} beziehungsweise t_v^{uv} zu klein, beziehungsweise zu groß gewählt werden, so dass das ILP möglicherweise nicht die Optimallösung zum aperiodischen Fahrplanproblem mit OD-Paaren liefert.

Die erste Idee, ein brauchbares M zu finden, besteht darin, M als die Summe der oberen Schranken U_a $\forall a \in \mathcal{A}$ zu setzen.

Lemma 3.2.23:
Für $M := \sum_{a \in \mathcal{A}} U_a$ löst $(ILP1)$ das aperiodische Fahrplanproblem mit OD-Paaren.

Beweis:
Wir müssen nur noch zeigen, dass $\Pi_j - \Pi_i \leq M \; \forall (i,j) \in \mathcal{E}(u) \times \mathcal{E}(u) \cup \mathcal{E}(v) \times \mathcal{E}(v)$ für jeden zulässigen Fahrplan Π gilt. Denn dann löst nach Lemma 3.2.22 $(ILP1)$ das aperiodische Fahrplanproblem mit OD-Paaren. Wenn $\Pi_l - \Pi_k \leq M \; \forall (i,j) \in \mathcal{E} \times \mathcal{E}$ erfüllt ist, gilt insbesondere $\Pi_j - \Pi_i \leq M \; \forall (i,j) \in \mathcal{E}(u) \times \mathcal{E}(u) \cup \mathcal{E}(v) \times \mathcal{E}(v)$. Wir zeigen hier $\Pi_l - \Pi_k \leq M \; \forall (k,l) \in \mathcal{E} \times \mathcal{E}$.
Sei also Π beliebiger zulässiger Fahrplan in $\mathcal{N}$. Es gibt nun die folgenden beiden Fälle:

1. Gibt es einen gerichteten Weg $P_0 = \{k, k+1, k+2, \ldots, l-1, l\}$ von k nach l, so gilt:

$$\begin{aligned}\Pi_l - \Pi_k &= \Pi_{k+1} - \Pi_k + \Pi_{k+2} - \Pi_{k+1} + \ldots + \Pi_l - \Pi_{l-1} \\ &\leq \underbrace{\Pi_{k+1} - \Pi_k}_{\leq U_{(k,k+1)}} + \underbrace{\Pi_{k+2} - \Pi_{k+1}}_{\leq U_{(k+1,k+2)}} + \ldots + \underbrace{\Pi_l - \Pi_{l-1}}_{\leq U_{(l-1,l)}} + \underbrace{\sum_{(p,q) \in \mathcal{A} \setminus P_0} (\Pi_q - \Pi_p)}_{\leq \sum_{(p,q) \in \mathcal{A} \setminus P_0} U_{(p,q)}} \\ &\leq \sum_{a \in \mathcal{A}} U_a = M \end{aligned} \tag{3.2}$$

2. Existiert kein gerichteter k-l-Weg, so gibt es aber einen ungerichteten k-l-Weg. Sei dieser $P = (k, k+1, k+2, \ldots, l-1, l)$. Dann gilt

$$\begin{aligned}\Pi_l - \Pi_k &\leq \underbrace{|\Pi_{k+1} - \Pi_k|}_{\leq U_{(k,k+1)}} + \underbrace{|\Pi_{k+2} - \Pi_{k+1}|}_{\leq U_{(k+1,k+2)}} + \ldots + \underbrace{|\Pi_l - \Pi_{l-1}|}_{\leq U_{(l-1,l)}} \\ &\quad + \underbrace{\sum_{(p,q) \notin P} (\Pi_q - \Pi_p)}_{\leq \sum_{(p,q) \in \mathcal{A} \setminus P} U_{(p,q)}} \\ &\leq \sum_{a \in \mathcal{A}} U_a = M \end{aligned} \tag{3.3}$$

Zusammen ergibt sich also $\Pi_l - \Pi_k \leq M \; \forall (k,l) \in \mathcal{E} \times \mathcal{E}$ und damit ist die Behauptung bewiesen.

□

Ein ähnlicher Beweis ist auch in der Dissertation von Marie Schmidt [Sch11] zu sehen.

Dieses M ist noch sehr groß und daher wird das ganzzahlige lineare Programm $(ILP1)$ auch für kleinere Instanzen langsam zu berechnen sein. Wir müssen nicht unbedingt für alle OD-Paare in den Bedingungen (14) und (15) das gleiche M nehmen, sondern können unterschiedliche M's berechnen (vgl. [Sch11]).

Definition 3.2.24:
M_u sei der optimale Zielfunktionswert des folgenden ganzzahligen linearen Programms:

$$
\begin{aligned}
(ILP2) \quad & \max \quad z && && (16)\\
& \text{s.d. } z \leq \Pi_j - \Pi_i + M(1 - d_{ij}) && \forall i,j \in \mathcal{E}(u) && (17)\\
& \sum_{i,j \in \mathcal{E}(u)} d_{ij} = 1 && && (18)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (19)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (20)\\
& d_{ij} \in \{0,1\} && \forall i,j \in \mathcal{E}(u) && (21)\\
& \Pi_i \in \mathbb{N} && \forall i \in \mathcal{E} && (22)
\end{aligned}
$$

und M_v der optimale Zielfunktionswert von

$$
\begin{aligned}
(ILP3) \quad & \max \quad z && && (23)\\
& \text{s.d. } z \leq \Pi_j - \Pi_i + M(1 - d_{ij}) && \forall i,j \in \mathcal{E}(v) && (24)\\
& \sum_{i,j \in \mathcal{E}(v)} d_{ij} = 1 && && (25)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (26)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (27)\\
& d_{ij} \in \{0,1\} && \forall i,j \in \mathcal{E}(v) && (28)\\
& \Pi_i \in \mathbb{N} && \forall i \in \mathcal{E} && (29)
\end{aligned}
$$

wobei $M = 2M_e^1$.

Lemma 3.2.25:
Das Programm

$$
\begin{aligned}
(ILP4) \quad \min \quad & \sum_{(u,v)\in OD} w_{uv}(t_v^{uv} - t_u^{uv}) && && (30)\\
\text{s. d.} \quad & \sum_{a\in\delta^{out}(u)} q_a^{uv} = 1 && \forall (u,v) \in OD && (31)\\
& \sum_{a\in\delta^{out}(e)} q_a^{uv} - \sum_{a\in\delta^{in}(e)} q_a^{uv} = 0 && \forall (u,v) \in OD && (32)\\
& \sum_{a\in\delta^{in}(v)} q_a^{uv} = 1 && \forall (u,v) \in OD && (33)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (34)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (35)\\
& t_u^{uv} \leq \Pi_i + M_u(1 - q_{(u,i)}^{uv}) && \forall (u,i) \in \mathcal{A}_{org} && (36)\\
& t_v^{uv} \geq \Pi_j - M_v(1 - q_{(j,v)}^{uv}) && \forall (j,v) \in \mathcal{A}_{dest} && (37)\\
& \Pi_i \in \mathbb{Z} && \forall i \in \mathcal{E} &&\\
& t_u^{uv}, t_v^{uv} \in \mathbb{Z} && \forall (u,v) \in OD &&\\
& q_a^{uv} \in \{0,1\} && \forall (u,v) \in OD, a \in \mathcal{A} &&
\end{aligned}
$$

mit M_u und M_v wie in Definition 3.2.24 löst das aperiodische Fahrplanproblem mit OD-Paaren und es gilt $M_u, M_v \leq M_e^1$.

Beweis:
$(ILP2)$ und $(ILP3)$ finden wegen den Timetabling-Bedingungen (19) und (20) beziehungsweise (26) und (27) jeweils einen zulässigen Fahrplan. Dieser berechnet wegen (17) beziehungsweise (24) und weil durch (18) und (21) beziehungsweise (25) und (28) genau ein Abfahrtsereignis $i \in \mathcal{E}(u)$ beziehungsweise Ankunftsereignis $j \in \mathcal{E}(u)$ ausgesucht wird

$$\max_{\Pi} \max_{i,j\in\mathcal{E}(u)} \Pi_j - \Pi_i$$

beziehungsweise

$$\max_{\Pi} \max_{i,j\in\mathcal{E}(v)} \Pi_j - \Pi_i.$$

Das heißt, dass für jeden zulässigen Fahrplan Π in $(ILP2)$ und für jedes OD-Paar $(u,v) \in OD$ die Relation $M_u = \max_{\Pi} \max_{i,j\in\mathcal{E}(u)} \Pi_j - \Pi_i \geq \Pi_j - \Pi_i$ gilt. Analog gilt auch in $(ILP3)$ für jeden zulässigen Fahrplan Π die Relation $M_v = \max_{\Pi} \max_{i,j\in\mathcal{E}(v)} \Pi_j - \Pi_i \geq \Pi_j - \Pi_i$.

□

Notation 3.2.26:
Ist $M_u = M$ und $M_v = M$ wie in Lemma 3.2.23 die Summe der oberen Schranken, so schreiben wir die M als M_e^1. Die Menge $M_u \times M_v$ aus Definition 3.2.24 schreiben wir als $M_e^2 = M_u \times M_v$ für jedes OD-Paar $(u, v) \in OD$.

Bemerkung 3.2.27:
Wenn wir die Zusammenhangskomponenten nicht durch jeweils eine Kante mit Gewicht $[0, 0]$ verbunden hätten, würde es dadurch, dass wir in $(ILP2)$ und $(ILP3)$ die Zielfunktion maximieren, möglich sein, zu große M_u beziehungsweise M_v zu berechnen.

Beispiel 15:
Wir betrachten Abbildung 3.3 mit OD-Paar $(u_1, v_1, 10)$. Wir berechnen $M_e^1 = 105$. Berechnen wir nun M_{u_1} und M_{v_1}, so erhalten wir $M_{u_1} = 105 = M_e^1$ und $M_{v_1} = 105 = M_e^1$. Fügen wir jedoch nun eine Kante $(1, 3)$ mit Gewicht $[0, 0]$ ein, so ergibt sich $M_{u_1} = 0$ und $M_{v_1} = 97$.

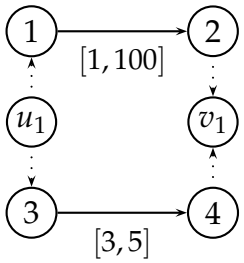


Abbildung 3.3: Ereignis-Aktivitätsnetzwerk mit OD-Paar $(u_1, v_2, 1)$

Eine weitere mögliche Methode um das Lösen des ganzzahligen linearen Programmes zu beschleunigen, ist eine untere Schranke für den optimalen Zielfunktionswert zu berechnen und dieses als Nebenbedingung mit in das ganzzahlige lineare Pro-

gramm zu schreiben. Sei LB diese untere Schranke, dann ergibt sich

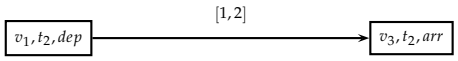
$$
\begin{aligned}
(ILP5) \quad \min \quad & \sum_{(u,v)\in OD} w_{uv}(t_v^{uv} - t_u^{uv}) && && (38)\\
\text{s. d.} \quad & \sum_{a\in\delta^{out}(u)} q_a^{uv} = 1 && \forall (u,v) \in OD && (39)\\
& \sum_{a\in\delta^{out}(e)} q_a^{uv} - \sum_{a\in\delta^{in}(e)} q_a^{uv} = 0 && \forall (u,v) \in OD && (40)\\
& \sum_{a\in\delta^{in}(v)} q_a^{uv} = 1 && \forall (u,v) \in OD && (41)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (42)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (43)\\
& t_u^{uv} \leq \Pi_i + M_u(1 - q_{(u,i)}^{uv}) && \forall (u,i) \in \mathcal{A}_{org} && (44)\\
& t_v^{uv} \geq \Pi_j - M_v(1 - q_{(j,v)}^{uv}) && \forall (j,v) \in \mathcal{A}_{dest} && (45)\\
& \sum_{(u,v)\in OD} w_{uv}(t_v^{uv} - t_u^{uv}) \geq LB && && (46)\\
& \Pi_i \in \mathbb{Z} && \forall i \in \mathcal{E} &&\\
& t_u^{uv}, t_v^{uv} \in \mathbb{Z} && \forall (u,v) \in OD &&\\
& q_a^{uv} \in \{0,1\} && \forall (u,v) \in OD, a \in \mathcal{A} &&
\end{aligned}
$$

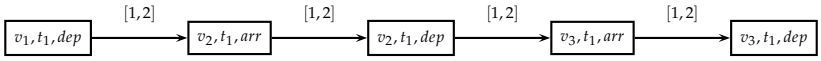
Wir werden später (in Kapitel 3.5) einige untere Schranken kennenlernen.

3.3 OD-Paare aus Abfahrts- und Ankunftsereignis

Wir nehmen nun an, dass wir anstelle der Menge der OD-Paare, die aus Paaren von Bahnhöfen besteht, eine Menge von OD-Paaren OD haben, die aus Abfahrts- und Ankunftsereignissen als Ursprung beziehungsweise Ziel besteht. Also gilt $OD = \{(i,j) \mid i \in \mathcal{E}_{dep}, j \in \mathcal{E}_{arr}\}$ wobei $i \in \mathcal{E}_{dep}$ die Abfahrt eines Zuges repräsentiert, die ein Passagier an einem bestimmten Bahnhof nehmen will und $j \in \mathcal{E}_{arr}$ die Ankunft eines Zuges, die ein Passagier an seinem Endbahnhof hat. Wiederum wird (i,j) ein Gewicht w_{ij} zugeordnet. (s. [SS10])

Beispiel 16:
Wir haben folgendes Netzwerk $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ gegeben:





Wir haben 5 Passagiere, die von v_1 nach v_3 wollen. Wenn wir jetzt wüssten, dass sie auf jeden Fall Abfahrtsereignis (v_1, t_2, dep) und (damit) Ankunftsereignis (v_3, t_2, arr) nehmen, dann könnten wir das OD-Paar schreiben als $((v_1, t_2, dep), (v_3, t_2, arr), 5)$.

Für jedes OD-Paar ist die Reisezeit $\Pi_v - \Pi_u$. Unser Ziel ist es wieder die gewichteten Summen der Reisezeiten von allen OD-Paaren zu minimieren:

$$\text{(P2)} \quad \min \sum_{\substack{(u,v)\in OD:\\ u\in\mathcal{E}_{dep}, v\in\mathcal{E}_{arr}}} w_{(u,v)}(\Pi_v - \Pi_u) \tag{47}$$

$$\text{s. d. } \Pi_j - \Pi_i \leq U_a \qquad \forall (i,j) \in \mathcal{A} \tag{48}$$

$$\Pi_j - \Pi_i \geq L_a \qquad \forall (i,j) \in \mathcal{A} \tag{49}$$

$$\Pi_i \in \mathbb{Z} \qquad \forall i \in \mathcal{E} \tag{50}$$

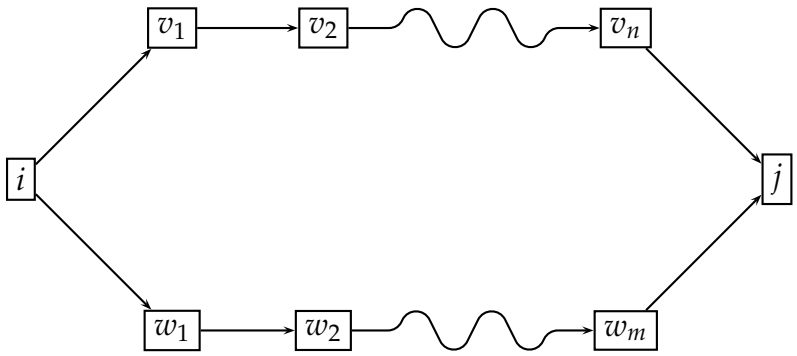
Satz 3.3.1: [SS10]
Das Problem (P2) kann durch lineare Programmierung gelöst werden.

Beweis:
Wie im Originalproblem sind die Zielfunktion und die Nebenbedingungen linear und die Matrix der Nebenbedingungen ist total unimodular, so dass das Problem mit linearer Programmierung gelöst werden kann.

□

Bemerkung 3.3.2: [SS10]
Die Reisezeit eines OD-Paars (i, j) hängt nur von der Zeit an Knoten i und Knoten j ab und nicht davon, welcher Weg von i nach j genommen wurde.

Beweis:
Sei $\mathcal{N}$ ein Ereignis-Aktivitätsnetzwerk mit OD-Paar (i, j), wobei $i \in \mathcal{E}_{dep}$ und $j \in \mathcal{E}_{arr}$. Seien weiterhin $P_1 = (i, v_1, v_2, \ldots, v_n, j)$ und $P_2 = (i, w_1, w_2, \ldots, w_m, j)$ zwei Wege von i nach j, wobei gilt $v_l \neq w_k$ für ein $l \in \mathbb{N}$ und ein $k \in \mathbb{N}$.



Dann bildet $C = P_1 \cup P_2$ einen ungerichteten Kreis in $\mathcal{N}$. Nach Lemma 2.4.16 gilt für jeden ungerichteten Kreis C:

$$tension(C) = 0\,.$$

Seien C^+ und C^- wie in Notation 2.4.15 die Vorwärts- beziehungsweise Rückwärtskanten von C. Nach Definition von *tension* gilt dann:

$$\begin{aligned} 0 &= tension(C) \\ &= \sum_{a=(k,l)\in C^+} \Pi_l - \Pi_k - \sum_{a=(k,l)\in C^-} \Pi_l - \Pi_k \\ &= \sum_{a=(k,l)\in P_1} \Pi_l - \Pi_k - \sum_{a=(k,l)\in P_2} \Pi_l - \Pi_k \end{aligned}$$

Damit folgt:

$$\sum_{a=(k,l)\in P_1} \Pi_l - \Pi_k = \sum_{a=(k,l)\in P_2} \Pi_l - \Pi_k$$

und die Länge von i nach j in P_1 ist dieselbe wie die Länge von i nach j in P_2. Damit hängt die Reisezeit nur von der Zeit an Knoten i und Knoten j ab und nicht vom gewählten Weg.

□

Ein gutes Mittel, um sich die Minimierung der gewichteten Summen der $\Pi_v - \Pi_u$ vorzustellen, ist die Einführung von virtuellen Kanten (s. [Sch11]), die für jedes OD-Paar (u, v) von u nach v gehen und die Gewichte $w_{(uv)}$ diesen zuzuordnen und $w_a = 0$ für alle anderen Kanten zu setzen. Wenn man das originale aperiodische Fahrplanproblem in diesem modifizierten Netzwerk löst, bekommt man die Formulierung (47)-(50).

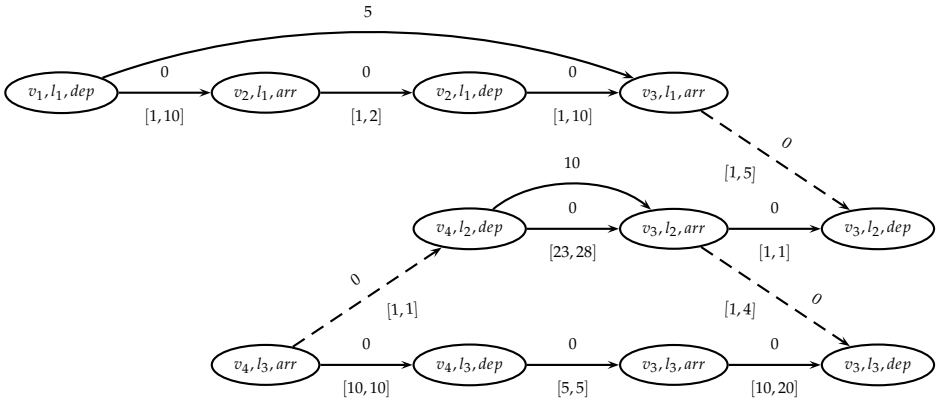
Beispiel 17:
Angenommen wir haben die OD-Paare

$$((v_1, l_1, dep), (v_3, l_1, arr), 5)$$

und

$$((v_4, l_2, dep), (v_3, l_2, arr), 10).$$

Dann sieht Beispiel 3 mit den oben genannten virtuellen Kanten so aus:



Lemma 3.3.3: [SS10]
Wenn wir für jedes OD-Paar (i, j) einen Weg P_{ij} von i nach j wählen und $w_a := \sum_{(i,j) \in OD : a \in P_{ij}} w_{ij}$ setzen, stimmt das aperiodische Fahrplanproblem ohne OD-Paare mit Gewichten w_a mit $(P2)$ überein.

Beweis:
Da die Zulässigkeitsbedingungen gleich sind, muss nur die Gleichheit der Zielfunktionswerte überprüft werden:

$$\begin{aligned}
\sum_{(k,l) \in \mathcal{A}} w_{(k,l)} \cdot (\Pi_l - \Pi_k) &= \sum_{(k,l) \in \mathcal{A}} \Big(\sum_{(i,j) \in OD : (k,l) \in P_{ij}} w_{ij} \Big) \cdot (\Pi_l - \Pi_k) \\
&= \sum_{(i,j) \in OD} \Big(\sum_{(k,l) \in P_{ij}} w_{ij} \Big) \cdot (\Pi_l - \Pi_k) \\
&= \sum_{(i,j) \in OD} w_{(i,j)} \cdot (\Pi_j - \Pi_i)
\end{aligned}$$

□

Es gilt:

Lemma 3.3.4: [SS10]
Das aperiodische Fahrplanproblem mit OD-Paaren kann durch das Lösen jeder ein-

zelnen Instanz $(\mathcal{N}, \widetilde{OD})$ des originalen Problems mit
$\widetilde{OD} := \{(e_{u_i}, e_{v_i}) : i = 1, \ldots, n\}$ mit $e_{u_i} \in \mathcal{E}(u_i), e_{v_i} \in \mathcal{E}(v_i)$ und durch das Vergleichen der Lösungswerte gelöst werden.

Bei beliebigen Kombinationen von (e_{u_i}, e_{v_i}) müssen wir dabei die Zulässigkeit kontrollieren, also herausfinden, ob es in $\mathcal{N}'$ einen Weg von e_{u_i} nach e_{v_i} gibt. Es gibt $\prod_{i=1}^{n} (|\mathcal{E}(u_i)| \cdot |\mathcal{E}(v_i)|)$ mögliche Kombinationen. Deshalb ist die Methode aus dem Lemma praktisch nicht durchführbar. Wenn es jedoch nur ein paar OD-Paare gibt, oder wenn die Ursprungs- und Zielknoten nur durch ein paar Abfahrts- und Ankunftsknoten verbunden sind, könnte es möglich sein die optimale Lösung doch so zu erhalten. Dies gilt zum Beispiel für den Fall, dass es für jedes OD-Paar (u, v) mit Bahnhöfen u und v nur jeweils ein mögliches Abfahrts- und Ankunftsereignis gibt.

Korollar 3.3.5: [SS10]
Wenn für jedes OD-Paar $(u_i, v_i) \in OD$ $|\mathcal{E}(u_i)| = |\mathcal{E}(v_i)| = 1$ gilt, kann die optimale Lösung für das aperiodische Fahrplanproblem mit OD-Paaren durch Lösen eines linearen Programms gefunden werden.

Auch im Fall, dass es nur ein OD-Paar gibt, ist das Problem "leicht" zu lösen.

Korollar 3.3.6: [SS10]
Wenn $OD = \{(u_i, v_i)\}$ gilt, kann die optimale Lösung für das aperiodische Fahrplanproblem mit OD-Paaren durch Lösen von höchstens $|\mathcal{E}(u_1)| \cdot |\mathcal{E}(v_1)|$ linearen Programmen gefunden werden.

3.3.1 Virtuelle-Kanten-basierende ILP-Formulierung

Wir haben bereits die Kanten-basierende ILP-Formulierung des aperiodischen Fahrplanproblems mit OD-Paaren kennengelernt. Da diese für Ereignis-Aktivitätsnetzwerke mit vielen Ereignissen eine sehr große Anzahl an Variablen benötigt, nutzen wir die in diesem Kapitel eingeführten virtuellen Kanten, um die Variablenanzahl in vielen Fällen zu verkleinern. Wir haben gesehen, dass die Fahrzeit für ein OD-Paar $(u_k, v_k) \in OD$ auf dem Weg $P = (u_k, i_1, i_2, \ldots, i_{u_k v_k}, v)$ nur vom ersten Abfahrtsereignis i_i und vom letzten Ankunftsereignis $i_{u_k v_k}$ abhängt (s. Bemerkung 3.3.2). Für jedes OD-Paar $(u_k, v_k) \in OD$ werden nun für alle Kombinationen von ersten Abfahrtsereignissen $i_1^{u_k v_k}$ und letzten Ankunftsereignissen $i_{u_k v_k}^{u_k v_k}$, für die es einen gerichteten Weg von $i_1^{u_k v_k}$ nach $i_{u_k v_k}^{u_k v_k}$ gibt, virtuelle Kanten gesetzt. Das ILP wählt dann für jedes OD-Paar genau eine Kombination aus erstem Abfahrtsereignis und letztem Ankunftsereignis aus (die, für die die Fahrzeit minimal ist).

Beispiel 18:

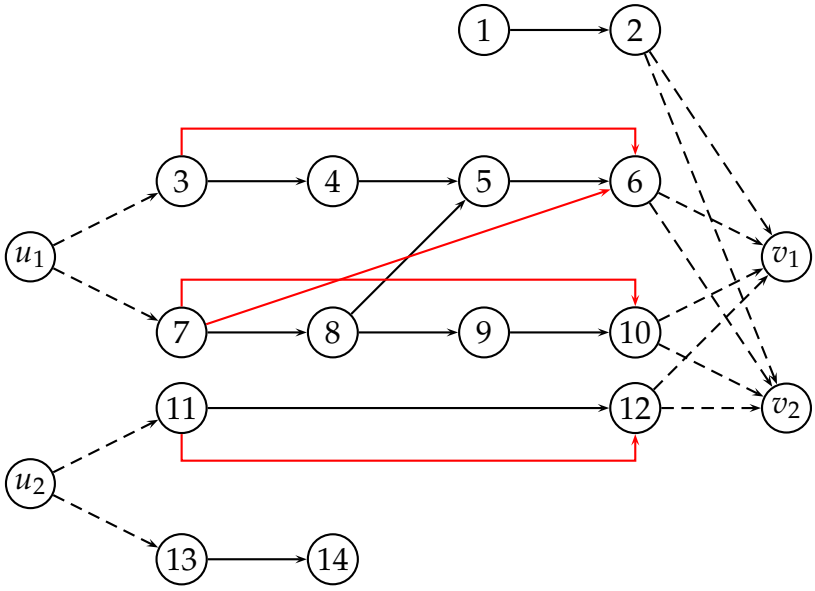


Abbildung 3.4: Ereignis-Aktivitätsnetzwerk mit OD-Paaren $(u_1, v_1), (u_2, v_2)$ und virtuellen Kanten $(3,6), (7,10), (7,6)$ und $(11,12)$

Zunächst formalisieren wir jedoch den Begriff der virtuellen Kanten:

Definition 3.3.7:
$\mathcal{A}^k_{\text{virt}} = \{(i^k_1, i^k_{u_k v_k}) \mid \exists \text{Weg } P = \{u_k, i^k_1, \ldots, i^k_{u_k v_k}, v_k\} \text{ in } \mathcal{N}' \text{ mit } i^k_1 \in \mathcal{E}_{dep}, i^k_{u_k v_k} \in \mathcal{E}_{arr}\}$
ist die Menge der virtuellen Kanten für OD-Paar $(u_k, v_k) \in OD$.

Wir führen die Variablen t^k wie folgt ein:
Für einen gegebenen Fahrplan Π ist

$$t^k := \min_{P^k_{ij}:(i,j)\in\mathcal{A}^k_{\text{virt}}} \Pi_j - \Pi_i$$

die minimale Reisezeit der Passagiere, die von u_k nach v_k reisen wollen. Weiterhin setzen wir:

$$z^k_{ij} = \begin{cases} 1 & \text{Virtuelle Kante } (i,j) \text{ wird von OD-Paar } (u_k, v_k) \text{ benutzt} \\ 0 & \text{sonst} \end{cases}$$

Das ILP [Sch11] ist dann:

$$
\begin{aligned}
(ILP6) \quad \min \quad & \sum_{(u_k,v_k)\in OD} w_k t^k && && (51)\\
\text{s.d. } & \Pi_j - \Pi_i \leq U_a && \forall a=(i,j)\in \tilde{\mathcal{A}} && (52)\\
& \Pi_j - \Pi_i \geq L_a && \forall a=(i,j)\in \tilde{\mathcal{A}} && (53)\\
& \sum_{P_{ij}^k:(i,j)\in\mathcal{A}_{\text{virt}}^k} z_{ij}^k = 1 && \forall (u_k,v_k)\in OD && (54)\\
& t^k \geq \Pi_j - \Pi_i - M^k(1-z_{ij}^k) && \forall (i,j)\in\mathcal{A}_{\text{virt}}^k, \forall (u_k,v_k)\in OD && (55)\\
& z_{ij}^k \in \{0,1\} && \forall (i,j)\in\mathcal{A}_{\text{virt}}^k, \forall (u_k,v_k)\in OD && (56)\\
& \Pi_i \in \mathbb{N} && \forall i\in\mathcal{E} && (57)\\
& t^k \in \mathbb{Z} && \forall (u_k,v_k)\in OD && (58)
\end{aligned}
$$

Die Zielfunktion (51) minimiert die Summe über die Fahrzeit gewichtet mit der Anzahl der Passagiere. (55) setzt die Fahrzeit für eine gewählte virtuelle Kante größer oder gleich der Zeit zwischen erstem Abfahrtsereignis und letztem Ankunftsereignis. Für ausreichend großes M^k ist (55) für die nicht gewählten virtuellen Kanten redundant. Da die Zielfunktion die Reisezeit minimiert, gilt für entsprechendes M^k somit $t^k = \Pi_j - \Pi_i \ \forall (i,j) \in \mathcal{A}_{\text{virt}}^k, \forall (u_k,v_k) \in OD$. Die Fahrzeit wird also genau auf die Differenz von erstem Abfahrtsereignis und letztem Ankunftsereignis gesetzt.

Lemma 3.3.8:
Der von $(ILP6)$ gefundene Fahrplan ist zulässig.

Beweis:
Nach Satz 2.4.19 ist ein Fahrplan genau dann zulässig, wenn

$$\Pi_j - \Pi_i \leq U_a \ \forall a=(i,j)\in\mathcal{A}$$

und

$$\Pi_j - \Pi_i \geq L_a \ \forall a=(i,j)\in\mathcal{A}\ .$$

Dies entspricht den Bedingungen (52) und (53), und der Fahrplan ist damit zulässig.

□

Lemma 3.3.9:
Für jedes OD-Paar $(u_k,v_k)\in OD$ wird ein u_k-v_k-Weg $P=(u_k,i_1^k,\ldots,i_{u_kv_k}^k)$ gefunden.

Beweis:
(54) und (56) sichern, dass für jedes OD-Paar $(u_k,v_k)\in OD$ genau eine virtuelle Kante $(i_1^k,i_{u_kv_k}^k)$ gewählt wird. Nach Definition der virtuellen Kanten wird für jedes OD-Paar (u_k,v_k) ein Weg in $\mathcal{N}'$ gefunden.

□

Wir definieren

Definition 3.3.10:

- $\mathcal{A}^+(u) := \{(u,i) \in \mathcal{A}_{org} \mid u \in \mathcal{E}_{org}\}$ ist die Menge der von Origin-Knoten u ausgehenden Origin-Kanten.
- $\mathcal{A}^-(v) := \{(j,v) \in \mathcal{A}_{dest} \mid v \in \mathcal{E}_{dest}\}$ ist die Menge der in Destination-Knoten v eingehenden Destination-Kanten.

Lemma 3.3.11:
$(ILP6)$ findet für genügend großes M^k, das heißt für

$$M^k \geq (\Pi_{j^k_{u_k v_k}} - \Pi_{j^k_1}) - (\Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1})$$

für alle zulässigen Fahrpläne Π, alle OD-Paare $(u_k, v_k) \in OD$ und alle virtuellen Kanten $(i^k_1, i^k_{u_k v_k}), (j^k_1, j^k_{u_k v_k}) \in \mathcal{A}^k_{\text{virt}}$ eine Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren.

Beweis:
Sei Π die Optimallösung, die $(ILP6)$ ausrechnet und z der entsprechende Zielfunktionswert. Angenommen es gibt eine bessere Lösung als die, die $(ILP6)$ ausrechnet. Diese Lösung habe den Zielfunktionswert z^*. Damit gelte also: $z^* < z$. Dann ist die Lösung gegeben durch einen Fahrplan Π^* und für jedes OD-Paar $(u_k, v_k) \in OD$ gibt es einen Weg $P(u_k, v_k)^* = (u_k, i^k_1, \ldots, i^k_{u_k v_k}, v_k)$. Wir konstruieren nun eine Lösung von $(ILP6)$, so dass diese Lösung

1. zulässig für $(ILP6)$ ist
2. Zielfunktionswert z^* hat.

Da die so konstruierte Lösung somit Zielfunktionswert $z = z^*$ hat, ist dies ein Widerspruch zu $z^* < z$ und die Behauptung wäre damit bewiesen.
Wir konstruieren nun die Lösung wie folgt. Wir setzen:

1. $\Pi = \Pi^*$
2. $z^k_{ij} = \begin{cases} 1 & \text{Der Weg } P(u_k, v_k)^* \text{ wird benutzt und} \\ & i, j \in P(u_k, v_k)^* \text{ mit } (u_k, i) \in \mathcal{A}^+(u_k), (j, v_k) \in \mathcal{A}^-(v_k) \\ 0 & \text{sonst} \end{cases}$
3. $t^k = \Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}$, wobei $\Pi_{i^k_1}$ der erste Abfahrtsknoten im Weg $P(u_k, v_k)^*$ und $\Pi_{i^k_{u_k v_k}}$ der letzte Ankunftsknoten ist.

Da Π^* zulässig ist, ist auch Π zulässig. Es gelten also $\Pi_j - \Pi_i \geq L_a$ und $\Pi_j - \Pi_i \leq U_a \ \forall a = (i,j) \in \mathcal{A}$. Das heißt, dass (52) und (53) erfüllt sind . (54) gilt, da für jedes OD-Paar (u_k, v_k) genau ein Weg ausgewählt wird und somit genau ein z_{ij}^k auf 1 gesetzt wird. Wir zeigen nun, dass Bedingung (55) erfüllt ist:
Sei $(i_1^k, i_{u_k v_k}^k)$ die benutzte virtuelle Kante für OD-Paar (u_k, v_k). Dann gilt:

$$t^k \geq \Pi_{i_{u_k v_k}^k} - \Pi_{i_1^k}$$

Seien $(l_1^k, l_{u_k v_k}^k)$ die nicht benutzten Kanten. Dann gilt:

$$z_{l_1^k l_{u_k v_k}^k}^k = 0 \quad (*)$$

Wegen $M^k \geq (\Pi_{j_1^k} - \Pi_{j_{u_k v_k}^k}) - (\Pi_{i_1^k} - \Pi_{i_{u_k v_k}^k}) \forall$ zulässigen Fahrpläne Π, alle OD-Paare $(u_k, v_k) \in OD$ und alle virtuellen Kanten $(i_1^k, i_{u_k v_k}^k), (j_1^k, j_{u_k v_k}^k) \in \mathcal{A}_{\text{virt}}^k$ gilt insbesondere

$$M^k \geq (\Pi_{l_{u_k v_k}^k} - \Pi_{l_1^k}) - (\Pi_{i_{u_k v_k}^k} - \Pi_{i_1^k}) \quad (**)$$

und damit

$$\begin{aligned}
\Pi_{l_{u_k v_k}^k} - \Pi_{l_1^k} - M^k (1 - z_{l_1^k l_{u_k v_k}^k}^k) &\overset{(*)}{=} \Pi_{l_{u_k v_k}^k} - \Pi_{l_1^k} - M^k \\
&\overset{(**)}{\leq} (\Pi_{l_{u_k v_k}^k} - \Pi_{l_1^k}) + (\Pi_{i_{u_k v_k}^k} - \Pi_{i_1^k}) - (\Pi_{l_{u_k v_k}^k} - \Pi_{l_1^k}) \\
&= \Pi_{i_{u_k v_k}^k} - \Pi_{i_1^k} \\
&\leq t^k \,. \qquad (3.4)
\end{aligned}$$

Es bleibt zu zeigen, dass der Zielfunktionswert von $(ILP6)$ mit der so konstruierten Lösung z^* beträgt und somit ein Widerspruch auftritt.
Sei dazu $l_{P(u_k,v_k)^*}^{\Pi^*}$ die Länge vom $u_k - v_k$-Weg $P(u_k, v_k)^*$ bezüglich Fahrplan Π^*. Dann gilt:

$$\begin{aligned}
z &< z^* \\
&= \sum_{(u_k,v_k) \in OD} w_{u_k v_k} \cdot l_{P(u_k,v_k)^*}^{\Pi^*} \\
&= \sum_{(u_k,v_k) \in OD} w_k \cdot (\Pi_{i_{u_k v_k}^k}^* - \Pi_{i_1^k}^*) \\
&= \sum_{(u_k,v_k) \in OD} w_k \cdot (\Pi_{i_{u_k v_k}^k} - \Pi_{i_1^k}) \\
&= \sum_{(u_k,v_k) \in OD} w_k t^k = z \quad \lightning
\end{aligned}$$

□

Marie Schmidt präsentiert in ihrer Dissertation [Sch11] einen ähnlichen Beweis.
Wir können nun folgendes Korollar formulieren:

Korollar 3.3.12:
Für $M^k \geq \Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}$ für alle zulässigen Fahrpläne, alle OD-Paare $(u_k, v_k) \in OD$ und alle virtuellen Kanten $(i^k_1, i^k_{u_k v_k}) \in \mathcal{A}^k_{\text{virt}}$ löst $(ILP6)$ das aperiodische Fahrplanproblem mit OD-Paaren.

Beweis:
Aus Lemma 3.3.11 wissen wir, dass $(ILP6)$ für $M^k \geq (\Pi_{j^k_1} - \Pi_{j^k_{u_k v_k}}) - (\Pi_{i^k_1} - \Pi_{i^k_{u_k v_k}})$ für alle zulässigen Fahrpläne Π, alle OD-Paare $(u_k, v_k) \in OD$ und alle virtuellen Kanten $(i^k_1, i^k_{u_k v_k}), (j^k_1, j^k_{u_k v_k}) \in \mathcal{A}^k_{\text{virt}}$ eine Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren findet. Da die Länge aller virtuellen Kanten für jeden Fahrplan positiv ist, gilt

$$(\Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}) - (\Pi_{j^k_{u_k v_k}} - \Pi_{j^k_1}) \leq (\Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}) \tag{3.5}$$

Somit löst $(ILP6)$ für $M^k \geq \Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}$ das aperiodische Fahrplanproblem mit OD-Paaren.

□

Welche M^k sind geeignet?

Auch hier stellt sich natürlich die Frage, wie man die M^k wählen kann, um $(ILP6)$ möglichst schnell zu lösen. Analog zu der Kanten-basierten Formulierung ist auch hier die erste Überlegung alle M^k auf die Summe der oberen Schranken zu setzen.

Lemma 3.3.13:
Für $M^k = \sum_{a \in \mathcal{A}} U_a$ für alle OD-Paare $(u_k, v_k) \in OD$ löst $(ILP6)$ das aperiodische Fahrplanproblem mit OD-Paaren.

Beweis:
Nach Korollar 3.3.12 müssen wir zeigen, dass $M^k = \sum_{a \in \mathcal{A}} U_a \geq \Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}$ für alle virtuellen Kanten $(i^k_1, i^k_{u_k v_k})$ und alle zulässigen Fahrpläne. Aus Lemma 3.2.23 wissen wir, dass $\sum_{a \in \mathcal{A}} U_a \geq \Pi_j - \Pi_i$ für alle $(i, j) \in \mathcal{E} \times \mathcal{E}$. Damit gilt also insbesondere auch $M^k = \sum_{a \in \mathcal{A}} U_a \geq \Pi_{i^k_{u_k v_k}} - \Pi_{i^k_1}$.

□

Der Beweis ist ähnlich zu dem, den Marie Schmidt in ihrer Dissertation [Sch11] zeigt.

Notation 3.3.14:
Wir schreiben die Menge der M^k als $M^1_v := M^k$.

Kleinere M^k bekommen wir, indem wir für jedes OD-Paar und jeden zulässigen Fahrplan die maximale Länge aller virtuellen Kanten bestimmen.

Definition 3.3.15: [Sch11]
$\widetilde{M^k}$ sei der optimale Zielfunktionswert des folgenden ganzzahligen linearen Programms:

$$
\begin{aligned}
(ILP7) \quad & \max \quad z && && (59)\\
& \text{s.d. } z \leq \Pi_j - \Pi_i + 2 \cdot M^k(1 - d_{ij}) && \forall (i,j) \in \mathcal{A}^k_{\text{virt}} && (60)\\
& \sum_{(i,j)\in\mathcal{A}^k_{\text{virt}}} d_{ij} = 1 && && (61)\\
& \Pi_j - \Pi_i \leq U_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (62)\\
& \Pi_j - \Pi_i \geq L_a && \forall a = (i,j) \in \tilde{\mathcal{A}} && (63)\\
& d_{ij} \in \{0,1\} && \forall (i,j) \in \mathcal{A}^k_{\text{virt}} && (64)\\
& \Pi_i \in \mathbb{R} && \forall i \in \mathcal{E} && (65)
\end{aligned}
$$

Bemerkung 3.3.16:
Im Gegensatz zu Bemerkung 3.2.27 ist es hier nicht notwendig, dass wir die Zusammenhangskomponenten mit Kanten mit Gewicht $[0,0]$ verbinden, da die Knoten Π_i und Π_j in Bedingung (60) auf Grund von $(i,j) \in \mathcal{A}^k_{\text{virt}}$ schon in einer Zusammenhangskomponente liegen. Um die Vergleichbarkeit der Fahrpläne zu gewährleisten, haben wir in unseren Tests in Kapitel 5.5, jedoch auch hier die Kanten $a \in \mathcal{A}_{con}$ eingefügt.

Lemma 3.3.17: [Sch11]
Mit $M^k = \widetilde{M^k}$ löst $(ILP8)$ das aperiodische Fahrplanproblem mit OD-Paaren.

Beweis:
Wegen (62) und (63) werden alle für $(ILP6)$ zulässigen Lösungen durchsucht. Durch (61) wird genau eine virtuelle Kante ausgewählt. Durch die Zielfunktion (59) und Bedingung (60) ist die Optimallösung von $(ILP8)$ $\max\limits_{\Pi} \max\limits_{(i,j)\in\mathcal{A}^k_{\text{virt}}} \Pi_j - \Pi_i$. Damit wird also für jedes OD-Paar die maximale Länge M^k aller virtuellen Kanten bestimmt und es gilt: $M^k = \max\limits_{\Pi} \max\limits_{(i,j)\in\mathcal{A}^k_{\text{virt}}} \Pi_j - \Pi_i \geq \Pi_j - \Pi_i$ für alle OD-Paare $(u_k, v_k) \in OD$, alle zulässigen Fahrpläne und alle virtuellen Kanten $(i,j) \in \mathcal{A}^k_{\text{virt}}$.

□

Auch hier können wir die Bedingung, dass der Zielfunktionswert größer als eine un-

tere Schranke ist, dazunehmen um das Finden der Optimallösung zu erleichtern:

$$
\begin{aligned}
(ILP8) \quad \min \quad & \sum_{(u_k,v_k)\in OD} w_k t^k && && (66)\\
\text{s.d. } & \Pi_j - \Pi_i \leq U_a && \forall a=(i,j)\in \tilde{\mathcal{A}} && (67)\\
& \Pi_j - \Pi_i \geq L_a && \forall a=(i,j)\in \tilde{\mathcal{A}} && (68)\\
& \sum_{P_{ij}^k:(i,j)\in\mathcal{A}_{\text{virt}}^k} z_{ij}^k = 1 && \forall (u_k,v_k)\in OD && (69)\\
& t^k \geq \Pi_j - \Pi_i - M^k(1-z_{ij}^k) && \forall (i,j)\in\mathcal{A}_{\text{virt}}^k, \forall (u_k,v_k)\in OD && (70)\\
& \sum_{(u_k,v_k)\in OD} w_k t^k \geq LB && && (71)\\
& z_{ij}^k \in \{0,1\} && \forall (i,j)\in\mathcal{A}_{\text{virt}}^k, \forall (u_k,v_k)\in OD && (72)\\
& \Pi_i \in \mathbb{N} && \forall i\in\mathcal{E} && (73)\\
& t^k \in \mathbb{Z} && \forall (u_k,v_k)\in OD && (74)
\end{aligned}
$$

Notation 3.3.18:
Wir schreiben die Menge der $\widetilde{M^k}$ als $M_v^2 := \widetilde{M^k}$.

Wir können das Problem jetzt mit ILP-Solvern (ganzzahlige lineare Solver) lösen. Da jedoch die Variablenanzahl sehr groß wird, ist das nur für kleine Instanzen ratsam. Dies motiviert die Idee eine Heuristik zu entwickeln, die das Problem hinreichend genau in polynomieller Zeit löst.

3.4 Heuristik

Wenn wir eine Möglichkeit finden, die OD-Menge, bestehend aus Bahnhöfen, in eine OD-Menge bestehend aus Ereignissen zu überführen, können wir mit den so berechneten Gewichten $w_a\ \forall a\in\mathcal{A}$ das Fahrplanproblem ohne OD-Paare lösen.

Eine Idee hierfür, die in Zusammenarbeit mit Marie Schmidt [Sch11] ausgearbeitet wurde, ist ein Verfahren, das im Netzwerk $\mathcal{N}'$ kürzeste Wege bezüglich Kantenlängen $K_a\ \forall a\in\mathcal{A}$ berechnet und dann diese Wege nutzt um das Fahrplanproblem (7) zu lösen und dieses Vorgehen iterativ zu wiederholen.

Wie setzt man nun dabei die Kantenlängen? Am Anfang bieten sich für die Kantenlängen $K_a\ \forall a\in\mathcal{A}$ zum Beispiel die unteren Schranken L_a der Kanten an. Aber auch die oberen Schranken U_a, $\frac{L_a+U_a}{2}$ oder ganz zufällige Werte können genommen werden. Diese brauchen noch keinen zulässigen Fahrplan ergeben.

Wenn wir die kürzesten Wege berechnet haben, wissen wir, welche Wege die Passagiere für diese Kantenlängen nehmen würden. Insbesondere kennen wir die Abfahrts- und Ankunftsereignisse. Wir können also die Gewichte $w_a\ \forall a\in\mathcal{A}$ bestimmen und durch lineare Programmierung einen für diese Wege optimalen Fahrplan berechnen (aperiodisches Fahrplanproblem ohne OD-Paare). Dieser ist nun zulässig.

Um die Wege zu verbessern führen wir das ganze noch einmal durch, indem wir die Kantenlängen K_a auf die Differenz $\Pi_j - \Pi_i$, die wir durch den berechneten Fahrplan haben, setzen, wiederum kürzeste Wege bezüglich der K_a berechnen und das aperiodische Fahrplanproblem ohne OD-Paare lösen. Wir iterieren diesen Prozes so lange bis wir in zwei aufeinanderfolgenden Iterationen den gleichen Fahrplan erhalten, denn dann wird sich bei einem deterministischen Kürzeste-Wege-Verfahren der Fahrplan nicht mehr ändern. Wir werden sehen, dass diese Heuristik in jedem Schritt gleich gut oder besser wird.

Algorithmus 3.4.1:

Input: Instanz I eines aperiodischen Fahrplanproblems mit OD-Paaren bestehend aus einem Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ und einer OD-Menge OD, Anzahl der Schritte S, Kantenlängen K_a

Output: Ein zulässiger Fahrplan für I beziehungsweise Abbruch, falls es keinen zulässigen Fahrplan gibt.

1: Leite die Passagiere durch $\mathcal{N}'$ mit einem Kürzeste-Wege-Algorithmus mit Kantenlängen K_a.
2: **if** $\nexists$ Weg für ein OD-Paar **or** $\exists$ Kreis negativer Länge im Netzwerk $\mathcal{N}^*$ **then**
3: STOP, die Instanz ist nicht zulässig
4: **end if**
5: Ordne jeder Kante in $\mathcal{N}$ ein Gewicht w_e^1 nach Zeile 1 zu, das die Anzahl der Passagiere repräsentiert.
6: Erstelle einen aperiodischen Fahrplan Π^1, der optimal ist, für die Kantengewichte w_e^1
7: **for** $i = 2$ **to** S **do**
8: Leite die Passagiere durch $\mathcal{N}'(\Pi^{i-1})$ mit Dijkstra's Algorithmus mit $\Pi_k^{i-1} - \Pi_j^{i-1}$ als Kantenlängen.
9: Ordne jeder Kante in $\mathcal{N}$ ein Gewicht w_e^i (aus Zeile 8) zu, das die Anzahl der Passagiere repräsentiert
10: Erstelle einen aperiodischen Fahrplan Π^i, der bzgl. der Gewichte w_e^i optimal ist.
11: **if** $\Pi^i = \Pi^{i-1}$ **or** max. Zeit ist überschritten **then**
12: return Π^i
13: **end if**
14: **end for**
15: return Π^S

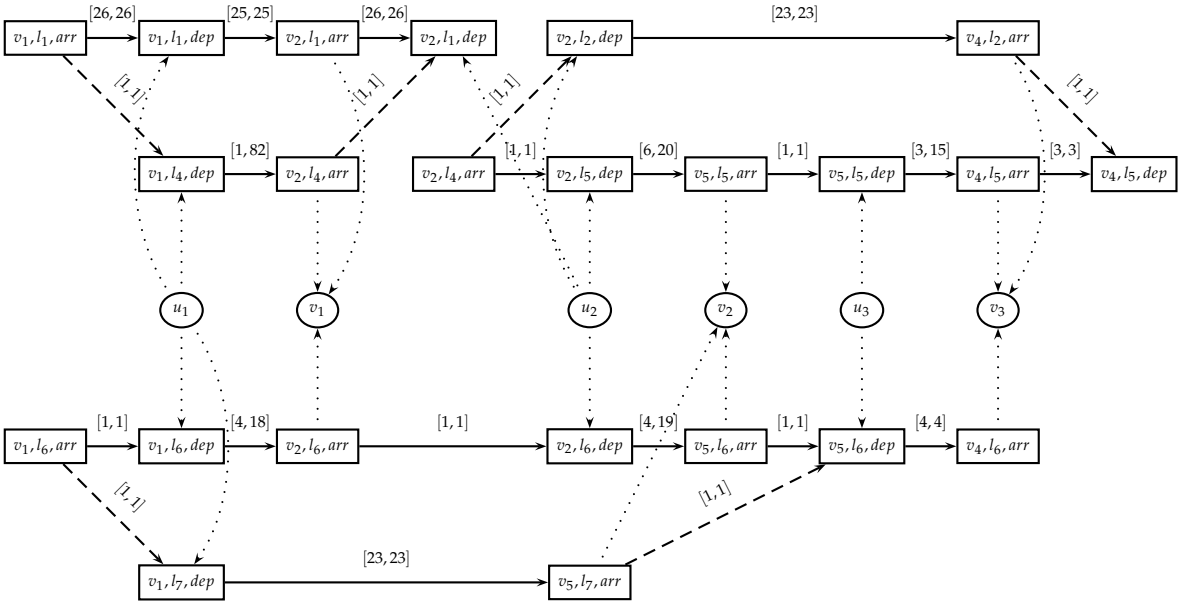
Bemerkung 3.4.2:
Das Erstellen der aperiodischen Fahrpläne in Zeile 6 und 10 wird zukünftig als Timetabling-Schritt bezeichnet.

Definition 3.4.3:

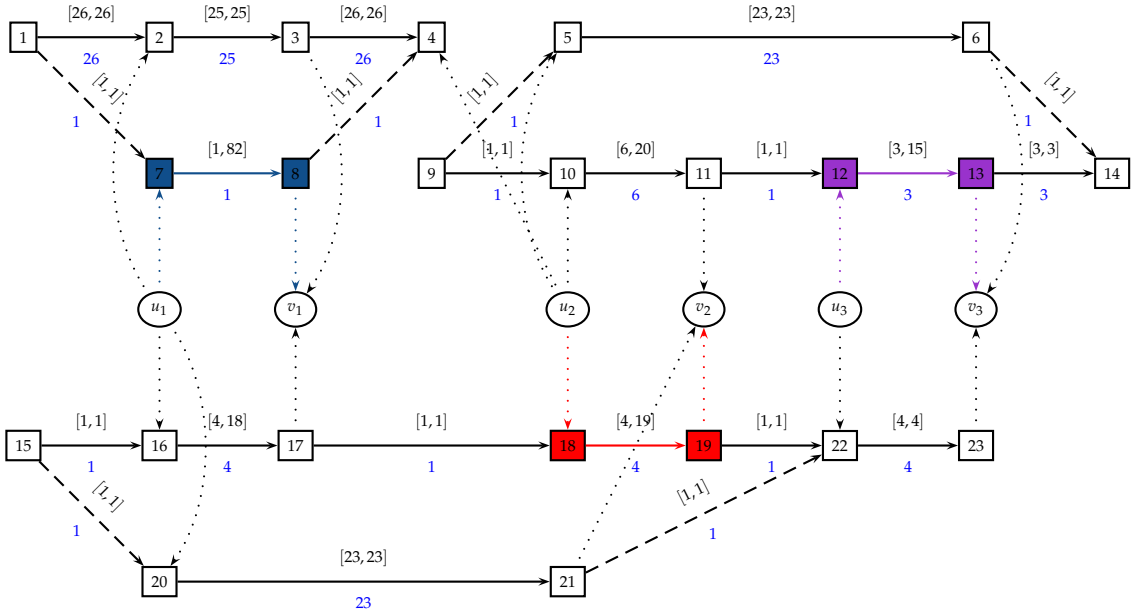
Ein Iterationsschritt der Heuristik besteht aus einem Routing und dem darauffolgenden Timetabling.

Wir stellen nun anhand eines Beispiels die Funktionsweise der Heuristik vor.

Beispiel 19:
Wir betrachten folgendes Beispiel mit $w_{u_1v_1} = 3$, $w_{u_2v_2} = 2$ und $w_{u_3v_3} = 1$.



Wir geben den Ereignissen eindeutige IDs. Markiert sind die kürzesten Wege der OD-Paare für Anfangskantenlängen L_a $\forall a \in \mathcal{A}$ und die OD-Paare.



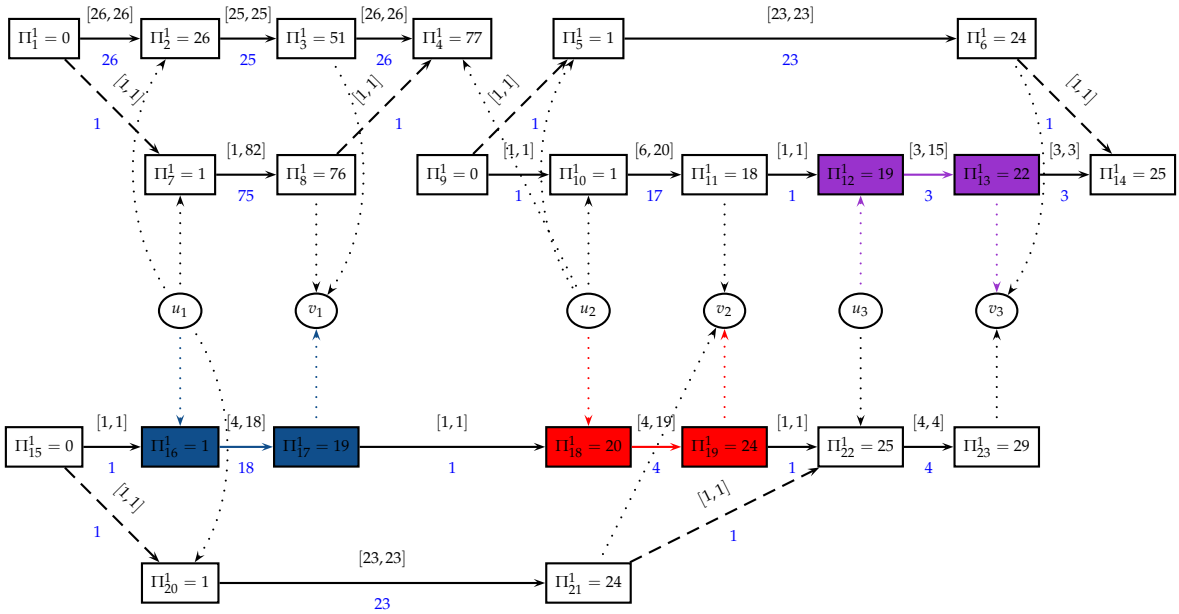
Lösen das lineare Programm:

$$\min 3 \cdot (\Pi^1_8 - \Pi^1_7) + 2 \cdot (\Pi^1_{19} - \Pi^1_{18}) + 1 \cdot (\Pi^1_{13} - \Pi^1_{12}) \tag{3.6}$$

$$\text{s.d. } \Pi^1_j - \Pi^1_i \leq U_a \qquad \forall a = (i,j) \in \mathcal{A} \tag{3.7}$$

$$\Pi^1_j - \Pi^1_i \geq L_a \qquad \forall a = (i,j) \in \mathcal{A}$$

Wir erhalten den Fahrplan Π^1 mit Zielfunktionswert 236, wodurch wir die Längen der Kanten als $\Pi^1_j - \Pi^1_i \ \forall (i,j) \in \mathcal{A}$ setzen können (markiert in blau) und folgende kürzeste Wege erhalten:



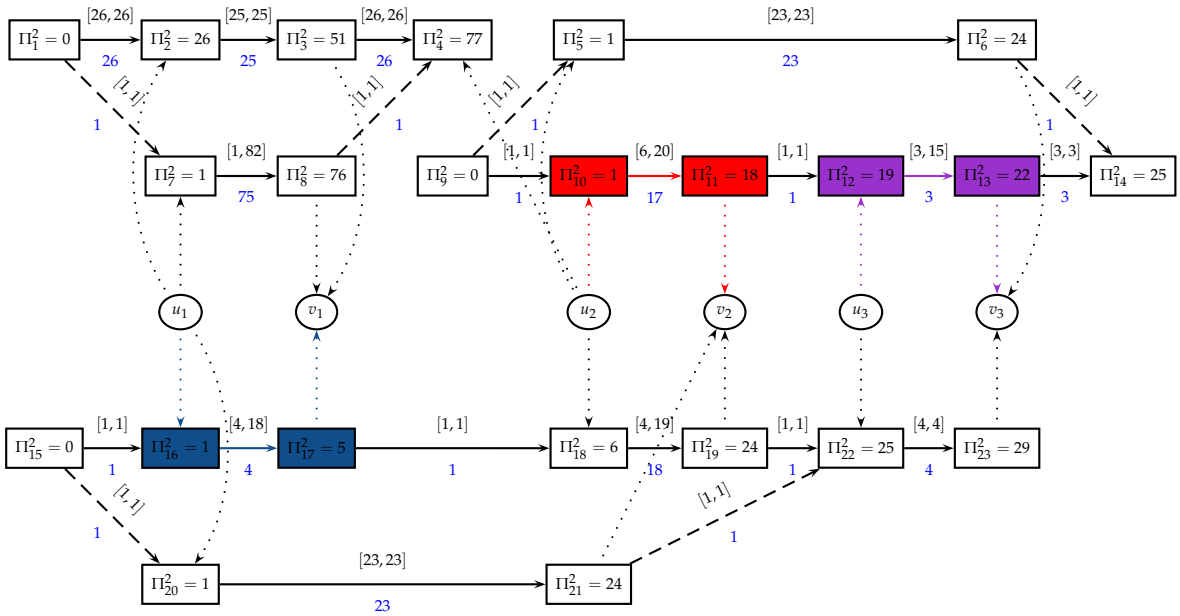
Wir lösen nun

$$\min 3 \cdot (\Pi^2_{17} - \Pi^2_{16}) + 2 \cdot (\Pi^2_{19} - \Pi^2_{18}) + 1 \cdot (\Pi^2_{13} - \Pi^2_{12}) \tag{3.8}$$

$$\text{s.d. } \Pi^2_j - \Pi^2_i \leq U_a \qquad \forall a = (i,j) \in \mathcal{A} \tag{3.9}$$

$$\Pi^2_j - \Pi^2_i \geq L_a \qquad \forall a = (i,j) \in \mathcal{A}$$

und erhalten einen neuen Fahrplan Π^2 mit Zielfunktionswert 51. Wir setzen wiederum neue Kantenlängen $\Pi^2_j - \Pi^2_i \ \forall (i,j) \in \mathcal{A}$ und können so kürzeste Wege bezüglich der neuen Längen bestimmen:



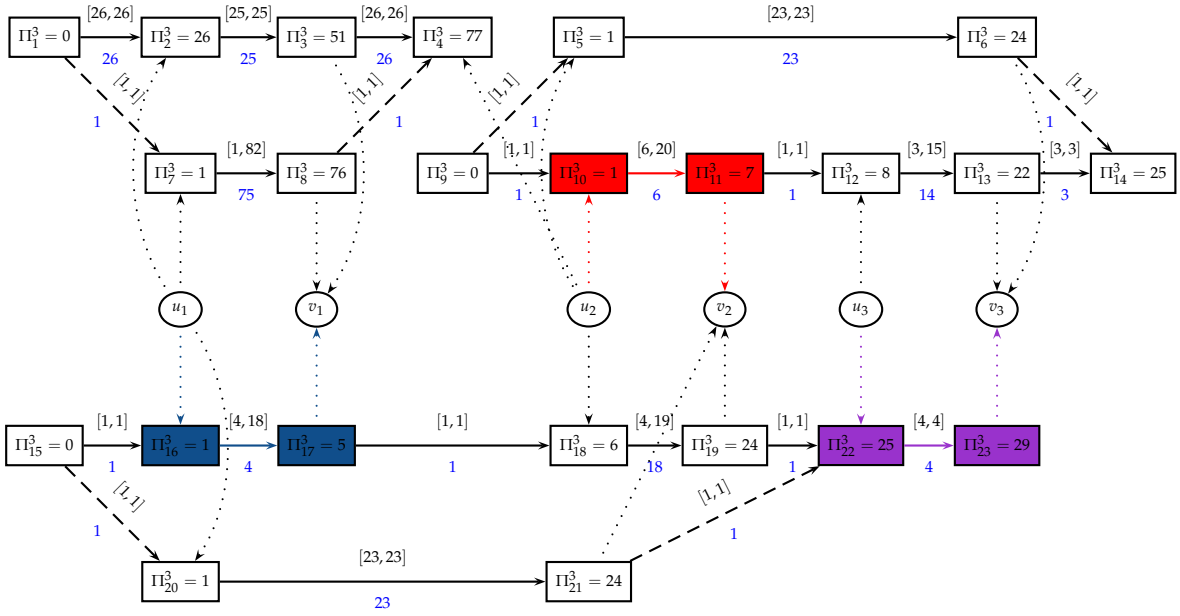
Wir lösen

$$\min 3 \cdot (\Pi^3_{17} - \Pi^3_{16}) + 2 \cdot (\Pi^3_{11} - \Pi^3_{10}) + 1 \cdot (\Pi^3_{13} - \Pi^3_{12}) \tag{3.10}$$

$$\text{s.d. } \Pi^3_j - \Pi^3_i \leq U_a \qquad \forall a = (i,j) \in \mathcal{A} \tag{3.11}$$

$$\Pi^3_j - \Pi^3_i \geq L_a \qquad \forall a = (i,j) \in \mathcal{A}$$

und erhalten einen Fahrplan Π^3 mit Zielfunktionswert 38.



Wir lösen

$$\begin{aligned} &\min 3 \cdot (\Pi^4_{17} - \Pi^4_{16}) + 2 \cdot (\Pi^4_{11} - \Pi^4_{10}) + 1 \cdot (\Pi^4_{23} - \Pi^4_{22}) \\ &\text{s.d. } \Pi^4_j - \Pi^4_i \leq U_a && \forall a = (i,j) \in \mathcal{A} \\ &\quad \Pi^4_j - \Pi^4_i \geq L_a && \forall a = (i,j) \in \mathcal{A} \end{aligned}$$

und erhalten einen Fahrplan Π^4 mit Zielfunktionswert 28. Dieser Fahrplan ist derselbe wie Π^3 aus der Iteration zuvor und deshalb bricht die Heuristik an dieser Stelle ab.

Definition 3.4.4:
Wir definieren

- $$RZ^{\Pi^k}_{w^j_a} := \sum_{a=(i,j)\in\mathcal{A}} w^j_a \cdot (\Pi^k_j - \Pi^k_i) = \sum_{a\in\mathcal{A}} w^j_a \cdot x^k_a,$$

  wobei x^k_a für $a \in \mathcal{A}$ die Länge der Aktivität a ist, als Reisezeit bezüglich Fahrplan Π^k in Iteration k und Passagierverteilung w^j_a in Iteration j

- $$RZ^{\Pi^0}_{w^1_a} = \sum_{a\in\mathcal{A}} w^1_a \cdot x^0_a$$

  ist die Reisezeit, die sich aus den Längen der Anfangslösung x^0_a und dem ersten Routen ergibt.

Bemerkung 3.4.5:
Für $j = k$ ist $RZ^{\Pi^k}_{w^k_a}$ der Zielfunktionswert der Heuristik aus Iteration k.

An Beispiel 19 konnte man Konvergenz der Zielfunktionswerte erkennen, was auch durch das folgende Lemma bestätigt wird.

Lemma 3.4.6:
Lässt man die Abbruchbedingung von Algorithmus 3.4.1 weg, so konvergieren die Zielfunktionswerte $RZ^{\Pi^k}_{w^k_a}$, die durch Algorithmus 3.4.1 iterativ berechnet werden, gegen eine Lösung. Insbesondere werden die Zielfunktionswerte in jeder Iteration nicht schlechter als in der Iteration davor.

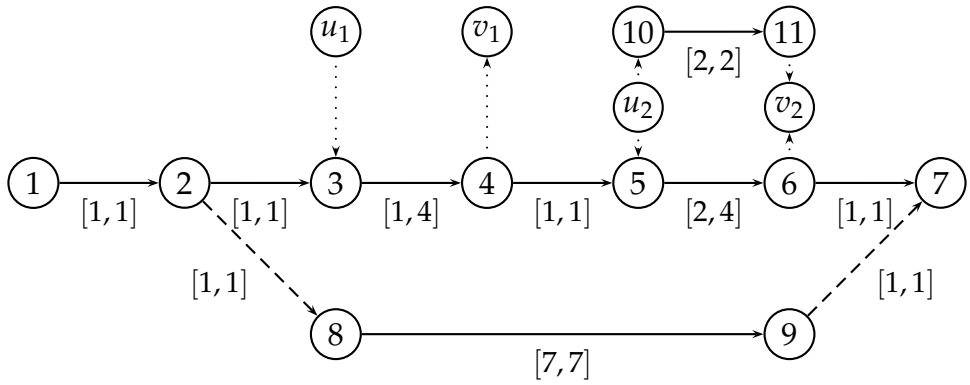
Beweis:
$RZ^{\Pi^k}_{w^j_a}$ sei wie in Definition 3.4.4 die Reisezeit bezüglich Passagerverteilung w^j_a in Iteration j und Fahrplan Π^k in Iteration k. Da das Timetabling in Iteration i einen für w^i_a optimalen Fahrplan liefert, wird in Iteration $i+1$ durch das Routing höchstens ein kürzerer Weg gewählt. Damit gilt:

$$RZ^{\Pi^i}_{w^i_a} \geq RZ^{\Pi^i}_{w^{i+1}_a} \geq RZ^{\Pi^{i+1}}_{w^{i+1}_a}$$

Mit vollständiger Induktion nach i und da die Zielfunktionswerte durch 0 von unten beschränkt sind, ergibt sich die Behauptung. □

Leider konvergiert Algorithmus 3.4.1 nicht immer gegen eine optimale Lösung, wie wir in Beispiel 20 sehen werden.

Beispiel 20:



Gegeben sind zwei OD-Paare $(u_1, v_1, 50)$ und $(u_2, v_2, 100)$. Der Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren ist 300. Wir führen die Heuristik mit den Startlängen L_a, U_a und $\frac{L_a+U_a}{2}$ und den beiden Kürzeste-Wege-Algorithmen `TreeMapQueue` und `FibonacciHeap` (s. Kapitel 4.2.5) aus und erhalten folgende Zielfunktionswerte:

	TMQ	FH
L_a	400	400
U_a	300	300
$\frac{L_a+U_a}{2}$	300	300

Alle Tests waren nach zwei Iterationen mit dem selben Zielfunktionswert für beide Iterationen fertig. Nimmt man als Startlängen die unteren Schranken $L_a \, \forall a \in \mathcal{A}$, so bekommt man nicht die optimale Lösung des aperiodischen Fahrplanproblems mit OD-Paaren.

Satz 3.4.7:
Der Algorithmus 3.4.1 arbeitet korrekt.

Beweis:
Zu zeigen ist, dass der Algorithmus als Output einen zulässigen Fahrplan ausgibt und dass er terminiert. Das Terminieren ist dadurch gesichert, dass wir eine endliche Anzahl S an Schritten mitgeben, nach der der Algorithmus abbricht. Es kann höchstens sein, dass der Abbruch schon nach einer Anzahl $r \leq S$ Schritten stattfindet. Dies ist der Fall, wenn $\Pi^i = \Pi^{i-1}$, wenn sich also der Fahrplan im Vergleich zu dem in der Iteration davor nicht verändert hat. Den Abbruch, falls kein zulässiger Fahrplan existiert, sichert Schritt 2 (vgl. Satz 2.4.19). Falls in diesem Schritt kein Abbruch erfolgt, kann also ein optimaler Fahrplan für die Gewichte w_e gefunden werden. Dieser

ist insbesondere zulässig. Da in jeder folgenden Iteration nur noch die Zielfunktion verändert wird und die Zulässigkeitsbedingungen gleich sind, bleibt der berechnete Fahrplan immer noch zulässig.

□

3.4.1 Fehlerabschätzung

Wir zeigen in diesem Kapitel anhand eines Beispiels, dass es nicht für alle Instanzen möglich ist, den relativen Fehler zu beschränken. Zunächst führen wir dazu den Begriff des relativen Fehlers ein.

Definition 3.4.8:
Sei $z^H_{K_a}$ der Zielfunktionswert, den die Heuristik mit Startlänge K_a bei Abbruch liefert und sei z^* die Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren. Dann ist der relative Fehler der Heuristik gegeben durch: $err_{rel} := \frac{z^H_{K_a} - z^*}{z^*}$

Lemma 3.4.9:
Für Startlänge $L_a \ \forall a \in \mathcal{A}$ ist es nicht immer möglich den relativen Fehler zu beschränken.

Beweis:
Wir betrachten:

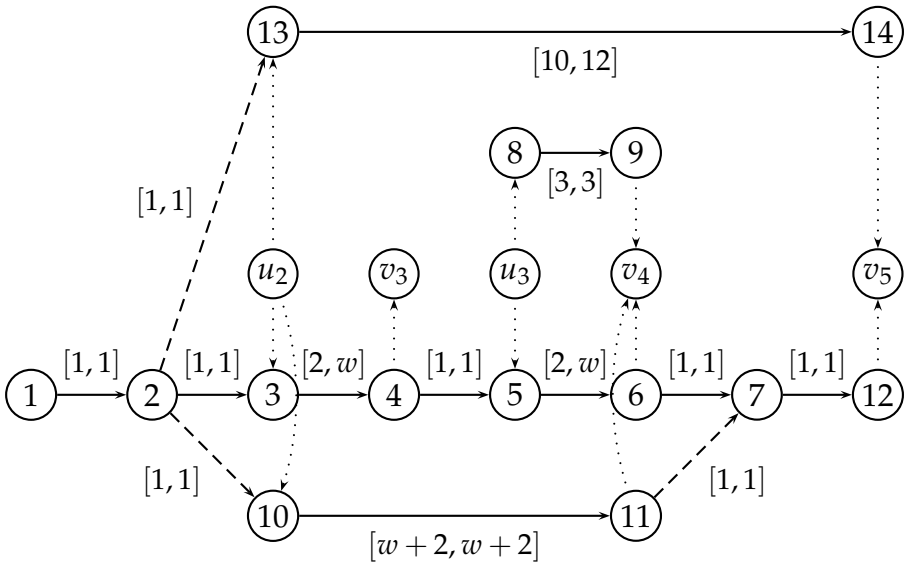


Abbildung 3.5: Der relative Fehler kann beliebig groß werden

mit den Ereignissen

$$\begin{aligned} 1 &= (dep, v_1) & 8 &= (dep, v_3) \\ 2 &= (arr, v_2) & 9 &= (arr, v_4) \\ 3 &= (dep, v_2) & 10 &= (dep, v_2) \\ 4 &= (arr, v_3) & 11 &= (arr, v_4) \\ 5 &= (dep, v_3) & 12 &= (arr, v_5) \\ 6 &= (arr, v_4) & 13 &= (dep, v_2) \\ 7 &= (dep, v_4) & 14 &= (arr, v_5) \end{aligned} \tag{3.12}$$

und den OD-Paaren $(u_2, v_3, 1)$, $(u_3, v_4, 3)$ und $(u_2, v_5, 1)$. Diese Instanz ist für $w \geq 5$ zulässig. Eine Optimallösung des aperiodischen Fahrplanproblems hat für $w > 8$ den Zielfunktionswert $z^* = 21$. Wenden wir die Heuristik auf diese Instanz mit Startlängen $L_a \ \forall a \in \mathcal{A}$ an, so ergibt sich für $w > 8$ ein Zielfunktionswert von $z^H_{L_a} = 1 \cdot (w-1) + 3 \cdot 2 + 1 \cdot 10 = (w-1) + 16$. Damit ergibt sich ein relativer Fehler von

$$err_{rel} = \frac{(w-1) + 16 - 21}{21} = \frac{w-6}{21} \xrightarrow{w \to \infty} \infty$$

Definition 3.4.10:
Wir definieren $diff_a := U_a - L_a$ als Differenz von oberer Schranke U_a und unterer Schranke L_a für Kante $a \in \mathcal{A}$.

Definition 3.4.11:
Sei z^* der optimale Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren und z^H der Zielfunktionswert, den die Heuristik bei Abbruch liefert. Dann definieren wir den absoluten Fehler der Heuristik durch

$$err_{abs} = z^H - z^*$$

Wir können für die Heuristik eine Abschätzung des absoluten Fehlers angeben.

Lemma 3.4.12:
Wenn als Startlänge $K_a = L_a$ gewählt wurde, gilt für den absoluten Fehler der Heuristik:

$$err_{abs} \leq \sum_{(u,v) \in OD} w_{(u,v)} \cdot diff_{(u,v)}$$

Dabei ist $diff_{(u,v)} = \sum_{a \in P(u,v)} diff_a$ für den Weg $P(u,v)$, der durch Algorithmus 3.4.1 gewählt wird.

Beweis:
Sei z^* der optimale Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren und $\tilde{z}$ der Lösungswert von Algorithmus 3.4.1. In der ersten Iteration wählt

Algorithmus 3.4.1 den kürzesten Weg für die OD-Paare (u,v) mit Längen $l_{(u,v)}$ bzgl. der unteren Schranken L_a als Dauer der Aktivitäten $a \in \mathcal{A}$. Sei $P_\Pi(u,v)$ ein kürzester Weg zwischen u und v in $\mathcal{N}(\Pi^*)$, wobei Π^* ein optimaler Fahrplan des aperiodischen Fahrplanproblems mit OD-Paaren ist. Dann gilt

$$l(P_\Pi(u,v)) \geq l_{(u,v)} \; .$$

Sei $\widetilde{\Pi}$ der Fahrplan, den Algorithmus 3.4.1 nach Iteration 1 berechnet hat. Im Netzwerk $\mathcal{N}(\widetilde{\Pi})$ mit $\widetilde{\Pi}$ als optimalem Fahrplan führt das Wählen der Wege, die optimal sind bzgl. der unteren Schranken, zu einer Lösung, wobei jeder Weg höchstens die Länge $\hat{l}_{(u,v)} = \sum_{a \in P(u,v)} U_a$ besitzt. Das heißt, nach Iteration 1 gilt:

$$\begin{aligned} err_{abs} &= \tilde{z} - z^* \\ &\leq \sum_{(u,v)\in OD} w_{(u,v)} \cdot (l(P_{\widetilde{\Pi}}(u,v)) - l(P_\Pi(u,v))) \\ &\leq \sum_{(u,v)\in OD} w_{(u,v)} \cdot (\hat{l}_{(u,v)} - l_{(u,v)}) \\ &\leq \sum_{(u,v)\in OD} w_{(u,v)} \cdot diff_{(u,v)} \end{aligned} \tag{3.13}$$

Da die Lösung des Algorithmus nach Lemma 3.4.6 in jedem Schritt besser wird, gilt dieses Ergebnis auch für den resultieren Fahrplan der Heuristik. □

3.5 Schranken für den Zielfunktionswert

Da wir nur in wenigen Fällen eine Optimallösung ausrechnen werden können, brauchen wir einen anderen Referenzwert um die durch Algorithmus 3.4.1 berechnete Lösung zu bewerten. Dafür bieten sich untere Schranken an. Die erste untere Schranke geben wir nur der Vollständigkeit halber an, werden sie im weiteren Verlauf jedoch nicht benutzen.

Lemma 3.5.1:
Sei $(LP - ILP1)$ die LP-Relaxation zu $(ILP1)$ und sei z_0 der Zielfunktionswert von $(LP - ILP1)$ und z^* der Zielfunktionswert von $(ILP1)$. Dann gilt:

$$LB_0 := z_0 \leq z^*$$

Beweis:
Da nach Satz 2.3.10 der Zielfunktionswert der LP-Relaxation eines ganzzahligen linearen Programms immer eine untere Schranke für den Zielfunktionswert des ganzzahligen linearen Programms liefert, folgt die Behauptung.

□

Wie auch schon in der Einleitung von Kapitel 3.4 geschrieben, müssen die Startlängen K_a für die Kanten $a \in \mathcal{A}$ nicht unbedingt einen zulässigen Fahrplan geben. Wenn die Kantenlängen jedoch einen zulässigen Fahrplan liefern und $K_a = L_a$ für alle $a = (i,j) \in \mathcal{A}$ (die Startlänge ist also für alle Kanten durch die unteren Schranken gegeben), so ist dies die Optimallösung. Daraus können wir die untere Schranke LB_1 für den Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren und somit auch für den Zielfunktionswert der Heuristik ableiten. Wir wissen nicht wieviele Passagiere Aktivität $a \in \mathcal{A}$ benutzen, deshalb bekommen wir eine untere Schranke für den Zielfunktionswert der Heuristik durch das Minimum aller Passagierzahlen über alle OD-Paare multipliziert mit dem Minimum aller unteren Schranken und der Anzahl der OD-Paare, wie folgender Satz aussagt:

Satz 3.5.2:
Seien w_{uv} die Anzahl der Passagiere, die von u nach v reisen wollen, und seien L_a die unteren Schranken des Spans $\Delta_a \forall a \in \mathcal{A}$. Sei m die Anzahl der OD-Paare. Dann gilt für den Zielfunktionswert z^* des aperiodischen Fahrplanproblems mit OD-Paaren:

$$LB_1 := m \cdot \min_{(u,v)\in OD} w_{uv} \min_{a\in\mathcal{A}} L_a \leq z^*$$

Beweis:
Die bestmögliche Länge einer Aktivität a ist die untere Schranke L_a, da $\Pi_j - \Pi_i \geq L_a \ \forall a = (i,j) \in \mathcal{A}$. Die kürzeste Reisezeit für ein OD-Paar (u,v) ist, wenn es über eine Kante geht, die das kleinste aller L_a als Länge hat. Es gilt:

$$z(\Pi) \geq m \cdot \min_{(u,v)\in OD} w_{uv} \min_{a\in\mathcal{A}} L_a$$

□

Eine weitere untere Schranke ist LB_2, die auch in der Dissertation von Marie Schmidt [Sch11] zu finden ist:

Satz 3.5.3:
Sei $SP_{L_a}(u,v)$ die Länge des kürzesten Weges von u nach v bezüglich der unteren Schranken $L_a \ \forall a \in \mathcal{A}$ in $\mathcal{N}$ und sei z^* Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Menge OD. Dann gilt:

$$LB_2 = \sum_{(u,v)\in OD} w_{uv} SP_{L_a}(u,v) \leq z^*.$$

Beweis:
Sei $P^*(u,v) = (u, i_1, \ldots, i_k, v)$ ein optimaler Weg von u nach v, den man durch das Lösen des aperiodischen Fahrplanproblems mit OD-Paaren erhält, Π^* der entsprechende Fahrplan und z^* der optimale Zielfunktionswert. Für einen zulässigen Fahrplan Π gilt $\Pi_{i_k} - \Pi_{i_1} \geq SP_{L_a}(u,v) \ \forall (u,v) \in OD \quad (*)$, wobei Π_{i_1} das erste Abfahrtser-

eignis und Π_{i_k} das letzte Ankunftsereignis für einen Weg von u nach v ist. Insbesondere gilt also:

$$LB_2 = \sum_{(u,v)\in OD} w_{uv} SP_{L_a}(u.v) \overset{(*)}{\leq} \sum_{(u,v)\in OD} w_{uv}(\Pi^*_{i_k} - \Pi^*_{i_1}) = z^*$$

□

Satz 3.5.4:
Sei $\mathcal{N}$ ein Ereignis-Aktivitätsnetzwerk mit OD-Menge $OD = \{(u_i, v_i) \mid i \in I\}$ und I eine Indexmenge. Sei (u_j, v_j) für ein $j \in I$ ein OD-Paar aus der OD-Menge OD. Sei $z^1(u_j, v_j)$ die optimale Reisezeit, die man durch das Lösen des aperiodischen Fahrplanproblems für Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ und dem einen OD-Paar $(u_j, v_j) \in OD$ erhält. Sei weiterhin $z^{OD}_{(\Pi)}(u_j, v_j)$ die Reisezeit von OD-Paar (u_j, v_j), die man durch das Lösen des aperiodischen Fahrplanproblems für Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ mit OD-Menge OD und Fahrplan Π erhält. Dann gilt für jeden Fahrplan Π:

$$z^1(u_j, v_j) \leq z^{OD}_{(\Pi)}(u_j, v_j)$$

Beweis:
Angenommen es gelte

$$z^1(u_j, v_j) > z^{OD}_{(\Pi)}(u_j, v_j) \ .$$

Dann ist $z^{OD}_{(\Pi)}(u_j, v_j)$ eine bessere Lösung für das aperiodische Fahrplanproblem mit OD-Paaren und $z^1(u_j, v_j)$ ist nicht optimal. Dies widerspricht der Definition von $z^1(u_j, v_j)$ und es muss also gelten

$$z^1(u_j, v_j) \leq z^{OD}_{(\Pi)}(u_j, v_j) \ .$$

□

Es ergibt sich folgendes Korollar:

Korollar 3.5.5:
Es gilt:

$$LB_3 := \sum_{(u,v)\in OD} w_{uv} z^1(u, v) \leq \sum_{(u,v)\in OD} w_{uv} z^{OD}(u, v) = z^* \ ,$$

wobei z^* der Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Menge OD ist.

Beweis:
Aus Lemma 3.5.4 wissen wir, dass gilt

$$z^1(u_j, v_j) \leq z^{OD}(u_j, v_j) \ .$$

Daraus folgt unmittelbar die Behauptung. □

Der Nachteil von LB_3 ist, dass man das aperiodische Fahrplanproblem für ein OD-Paar oft (genauer: Anzahl der OD-Paare mal) lösen muss. Wie wir jedoch gleich sehen, ist LB_3 die größte unserer untersuchten Schranken und kommt damit der Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren am nächsten.
Wie stehen diese Schranken also in Relation zueinander? Wir vergleichen zunächst LB_1 und LB_2 und erhalten folgendes Lemma:

Lemma 3.5.6:
Es gilt:

$$LB_1 \leq LB_2 \, .$$

Beweis:
Wir wissen, dass $w_{u_j v_j} \geq \min\limits_{(u,v)\in OD} w_{uv} \; \forall (u_j, v_j) \in OD$ gilt. Wenn man das aperiodische Fahrplanproblem mit OD-Paaren mit einem OD-Paar $(u, v) \in OD$ löst, wird die Reisezeit mindestens das Minimum aller unteren Schranken L_a annehmen. Dieser Fall tritt dann ein, wenn es einen Weg von u nach v gibt, der genau die Länge $\min\limits_{a\in\mathcal{A}} L_a$ hat, also nur über eine Kante mit der Länge $\min\limits_{a\in\mathcal{A}} L_a$ geht. Ist dies nicht der Fall, ist der kürzeste Weg von u nach v bezüglich der unteren Schranken größer. Es gilt also $SP_{L_a}(u, v) \geq \min\limits_{a\in\mathcal{A}} L_a$. Zusammen erhalten wir:

$$\begin{aligned} LB_1 &= m \cdot \min_{(u,v)\in OD} w_{uv} \min_{a\in\mathcal{A}} L_a \\ &= \sum_{n=1}^{m} \min_{(u,v)\in OD} w_{u,v} \min_{a\in\mathcal{A}} L_a \\ &\leq \sum_{(u,v)\in OD} w_{uv} SP_{L_a}(u, v) = LB_2 \end{aligned} \tag{3.14}$$

□

Es gilt weiterhin:

Lemma 3.5.7:
Es gilt:

$$LB_2 \leq LB_3 \, .$$

Beweis:
$z^1(u, v)$ ist die Reisezeit, die wir durch das Lösen des aperiodischen Fahrplanproblems mit OD-Paar $(u, v) \in OD$ erhalten. Nach Definition des Problems wird ein kürzester Weg P von u nach v gefunden (sofern es einen Weg von u nach v gibt),

sowie ein zulässiger Fahrplan Π. Wir zeigen nun, dass $z^1(u,v) \geq SP_{L_a}(u,v)$ ist:

$$\begin{aligned} z^1(u,v) &= \sum_{a=(i,j)\in P} \Pi_j - \Pi_i \\ &\overset{(*)}{\geq} \sum_{a=(i,j)\in P} L_a \\ &\geq \min_{\substack{P' \text{ Weg von} \\ u \text{ nach } v}} \sum_{a=(i,j)\in P'} L_a \\ &= SP_{L_a}(u,v) \end{aligned} \tag{3.15}$$

Dabei gilt $(*)$, da Π zulässig und damit $\Pi_j - \Pi_i \geq L_a \ \forall a = (i,j) \in \mathcal{A}$ ist. Damit erhalten wir:

$$LB_2 = \sum_{(u,v)\in OD} w_{u,v} SP_{L_a}(u,v) \leq \sum_{(u,v)\in OD} w_{u,v} z^1(u,v) = LB_3$$

□

Aus Lemma 3.5.6 und Lemma 3.5.7 erhalten wir somit unmittelbar:

Lemma 3.5.8:
Es gilt:

$$LB_1 \leq LB_3$$

Auch einige obere Schranken können wir finden:

Lemma 3.5.9:
Sei $(RZ^{\Pi^0}_{w_a^1})^{U_a}$ wie in Definition 3.4.4 die Reisezeit nach dem ersten Routen und vor dem ersten Timetabling bezüglich Startlängen $U_a \ \forall a \in \mathcal{A}$. Sei weiterhin z^* der optimale Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren. Dann gilt:

$$UB_1 := (RZ^{\Pi^0}_{w_a^1})^{U_a} \geq z^*$$

Beweis:
Die gesamte Fahrzeit bezüglich der oberen Schranken kann sich nach dem ersten Timetabling-Schritt niemals verschlechtern. Wenn es nach dem ersten Timetabling-Schritt keine Änderung der Fahrpläne mehr gibt und die Startlängen $U_a \ \forall a \in \mathcal{A}$ sind, so ist dies die Optimallösung. Da sich auch in jeder Iteration die Zielfunktionswerte niemals verschlechtern, ist $(RZ^{\Pi^0}_{w_a^1})^{U_a}$ eine obere Schranke für den optimalen Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren.

□

Wie wir bereits gesehen haben, besagt Lemma 3.4.6, dass die Zielfunktionswerte, die die Heuristik liefert, konvergieren. Das heißt, wir bekommen nach jeder Iteration eine obere Schranke für das aperiodische Fahrplanproblem mit OD-Paaren. Insbesondere ist also auch der Zielfunktionswert, den die Heuristik bei Abbruch liefert, eine obere Schranke.

Lemma 3.5.10:
Sei z^H der Zielfunktionswert, den die Heuristik beim Abbruch liefert. Sei z^* der optimale Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren. Dann gilt:

$$UB_2 := z^H \geq z^*$$

Beweis:
Da sich in jeder Iteration die Zielfunktionswerte niemals verschlechtern, ist z^H eine obere Schranke für den optimalen Zielfunktionswert des aperiodischen Fahrplanproblems mit OD-Paaren.

□

Korollar 3.5.11:
Es gilt: $UB_1 \geq UB_2$.

3.6 Abbruch der Heuristik

Für die Funktionsweise der Heuristik gibt es die folgenden vier möglichen Fälle:

1. Die Heuristik bricht nach einer endlichen Anzahl an Iterationen $< S$ ab und liefert in jeder Iteration den gleichen Zielfunktionswert.

2. Die Heuristik bricht nach einer endlichen Anzahl an Iterationen $< S$ ab und liefert unterschiedliche Zielfunktionswerte (nicht notwendigerweise in jeder Iteration).

3. Die Heuristik bricht erst nach S Iterationen ab und liefert immer den gleichen Zielfunktionswert.

4. Die Heuristik bricht erst nach S Iterationen ab und liefert unterschiedliche Zielfunktionswerte.

Je nach Implementation des Kürzeste-Wege-Algorithmus kann es passieren, dass die Heuristik in einem lokalen Optimum landet, aus der sie nicht mehr herausfindet. Dieses nennen wir einen Zykel.

Definition 3.6.1:
Wenn sich der Fahrplan in verschiedenen Iterationen i und j mit $i < j$ und $j \neq i + 1$ periodisch wiederholt, so nennt man dies einen Zykel.

Wir können die Ergebnisse von Lemma 3.5.3 ausnutzen, um uns folgendes Lemma zu überlegen.

Lemma 3.6.2:
Falls die Startlängen die unteren Schranken $L_a \; \forall a \in \mathcal{A}$ sind und diese einen zulässigen Fahrplan liefern, dann ist dieser ein optimaler Fahrplan.

Beweis:
Für den Fall, dass die L_a einen zulässigen Fahrplan liefern, gilt $\Pi_j - \Pi_i = L_a \ \forall a = (i,j) \in \mathcal{A}$. Die Heuristik wählt sich einen kürzesten Weg $P = (u, i_1, i_2, \ldots, i_n, v)$ von u nach v. Da der Fahrplan Π zulässig ist bezüglich L_a, gilt insbesondere: $SP_{L_a}(u,v) = \Pi_{i_1} - \Pi_u + \Pi_{i_2} - \Pi_{i_1} + \ldots + \Pi_v - \Pi_{i_n} = \Pi_v - \Pi_u$. Damit gilt

$$z_3 = \sum_{(u,v)\in OD} w_{uv} SP_{L_a}(u,v) = z^*.$$

□

Wie kann man jedoch überprüfen, ob man aus gegebenen Fahrzeiten $T_a \ \forall a \in \mathcal{A}$ einen zulässigen Fahrplan rekonstruieren kann?

Algorithmus 3.6.3:
Input: Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ mit Kantenlängen $T_a \ \forall a \in \mathcal{A}$
Output: Ein bzgl. $\mathcal{N}$ zulässiger Fahrplan oder Abbruch, wenn dies nicht möglich ist

Setze $\Pi_j = \infty \ \forall j$
for $i \in$ Zusammenhangskomponenten in $\mathcal{N}$ **do**
  Suche ein j mit $|\delta^-(j)| = 0$
  Setze $\Pi_j = 0$
  Setze $R = \emptyset$
end for
for $j \in \{\mathcal{E} \mid \Pi_j \neq \infty\}$ **do**
  if $R \neq \mathcal{E}$ **and** $|\delta^+(j)| \neq 0$ **then**
    for $k \in \mathcal{E}$ **do**
      for $l \in \mathcal{A}$ **do**
        if $\Pi_k == \infty$ **and** $\alpha(l) = j$ **and** $\omega(l) = k$ **then**
          Setze $\Pi_k = \Pi_j + T_a$
          $R = R \cup j$
        else if $\Pi_k \neq \infty$ **and** $\alpha(l) = j$ **and** $\omega(l) = k$ **then**
          if $\Pi_k \neq \Pi_j + T_a$ **then**
            Es kann kein zulässiger Fahrplan konstruiert werden.
          end if
        end if
      end for
    end for
  end if
end for
for $a = (i,j) \in \mathcal{A}$ **do**
  if $\Pi_j - \Pi_i \geq U_A$ **or** $\Pi_j - \Pi_i \leq L_a$ **then**
    Der konstruierte Fahrplan ist nicht zulässig.
  end if
end for
return Π

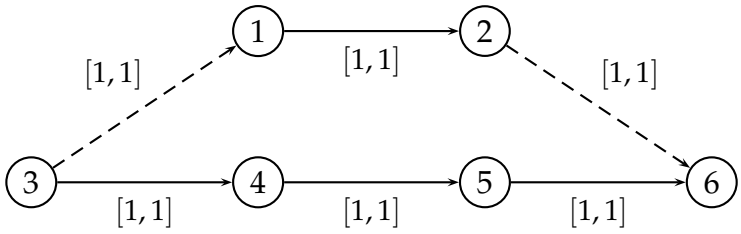
Lemma 3.6.4:
Algorithmus 3.6.3 arbeitet korrekt.

Beweis:
Der Algorithmus terminiert, da die Anzahl der Ereignisse endlich ist. Die Zeilen $23-27$ fragen ab, ob $L_a \leq \Pi_j - \Pi_i \leq U_a \ \forall a = (i,j) \in \mathcal{A}$ und sichern damit, dass falls ein Fahrplan Π aus den Startlängen T_a konstruiert werden konnte, dieser bezüglich dem Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ zulässig ist. Da in den Zeilen $15-17$ nur dann kein Abbruch geschieht, wenn $\Pi_k = \Pi_j + T_a$, gilt für einen gefundenen Fahrplan Π insbesondere $\Pi_j - \Pi_i = T_a$.

□

Bemerkungen 1:

1. Wäre die Abfrage in den Zeilen $23-27$ nicht gegeben, so könnte zum Beispiel aus dem Ereignis-Aktivitätsnetzwerk $\mathcal{N}$



   mit $T_a = 5 \ \forall a \in \mathcal{A}$ der Fahrplan

$$\Pi_1 = 5 \qquad \Pi_4 = 5 \tag{3.16}$$
$$\Pi_2 = 10 \qquad \Pi_5 = 10 \tag{3.17}$$
$$\Pi_3 = 0 \qquad \Pi_6 = 15 \tag{3.18}$$

   konstruiert werden. Es gilt jedoch zum Beispiel $\Pi_2 - \Pi_1 = 5 \notin [1,1]$. Damit ist der konstruierte Fahrplan nicht zulässig für das Ereignis-Aktivitätsnetzwerk $\mathcal{N}$.

2. Wenn Algorithmus 3.6.3 keinen zulässigen Fahrplan findet, sagt dies nichts darüber aus, ob es nicht doch vielleicht einen zulässigen Fahrplan gibt.

3. Ist $T_a \in [L_a, U_a] \ \forall a \in \mathcal{A}$ und ein Fahrplan kann konstruiert werden, so ist dieser zulässig.

Man kann nun mit Hilfe von Lemma 3.6.2 eine Abbruchbedingung zu Algorithmus 3.4.1 hinzufügen, die vor dem ersten Routen überprüft, ob die Startlängen $L_a \ \forall a \in \mathcal{A}$ sind. In diesem Fall wird mit Algorithmus 3.6.3 versucht, einen zulässigen Fahrplan zu rekonstruieren. Im Erfolgsfall kann dann die Heuristik nach dem ersten Routen abgebrochen werden und der entsprechende Fahrplan Π mit dem Zielfunktionswert,

der sich aus der Verteilung der Passagiere vom Routen und dem Fahrplan Π ergibt, kann zurückgegeben werden. Bei unseren praktischen Tests (s. auch Kapitel 5) haben wir dieses Kriterium jedoch nicht angewandt.

Eine weitere Abbruchbedingung ist der Fall, dass ein Zykel existiert. Wenn wir also alle Fahrpläne speichern und den gerade gefundenen mit allen Fahrplänen aus vorherigen Iterationen vergleichen, kann es sein, dass wir auch hier eine Übereinstimmung feststellen können. Da ein deterministisches Kürzeste-Wege-Verfahren wieder genau dieselben Fahrpläne liefert wie im Zyklus davor (also zwischen erstem und letztem Auftreten des gleichen Fahrplans), kann auch hier die Zeit, die die Heuristik braucht, verkürzt werden. Wie wir in Kapitel 5 sehen, werden wir dieses Kriterium auch nicht verwenden.

Wir werden für unsere Tests nur die in Algorithmus 3.4.1 in Zeile 11 beschriebenen Bedingungen und die vorgebene maximale Anzahl an Iterationen zum Abbruch benutzen. Das heißt, wir überprüfen, ob ein vorgegebenes Zeitlimit überschritten ist, ob zwei Fahrpläne in zwei aufeinanderfolgende Iterationen übereinstimmen oder ob die maximale Anzahl an Iterationen erreicht beziehungsweise überschritten ist.

Teil II

Praktischer Teil

4 Implementierung der Heuristik

Wir werden in diesem Kapitel auf alles eingehen, was programmiertechnisch erarbeitet wurde. Dazu werden wir einige selbstentwickelte Programme vorstellen, die zur Erleichterung des Arbeitens mit der Heuristik dienen. Wir präsentieren ein paar schon vorhandene Hilfsmittel, die wir benutzt haben. Das Hauptaugenmerk liegt schließlich beim Vorstellen der Implementation der Heuristik.

4.1 Einführung

Wir geben zunächst einen kurzen Überblick über die implementierten Programme:

- ExactMethod - Die exakte Methode $(ILP4)/(ILP5)$ mit M_e^1 und M_e^2 beziehungsweise $(ILP6)/(ILP8)$ mit M_v^1 und M_v^2
- Heuristik - Algorithmus 3.4.1
- MakeGoodODs - Aus der Basis-OD-Datei, die aus LinTim bekannt ist, die OD-Paare (u, v) (symmetrisch, also auch für (v, u)) auswählen, für die es einen Weg von u nach v gibt.
- ChooseODs - Aus einer Basis-OD-Datei eine bestimmte oder zufällige Anzahl an OD-Paaren auswählen
- PlotFiles - Programm um ein Bash-Script zum Plotten der Zielfunktionswerte aus allen Zielfunktionswertdateien im angegebenen Verzeichnis zu erstellen
- SelectPartEAN - Programm um eine .dot-Datei zum Zeichnen des EANs $\mathcal{N}'$ zu erstellen
- GeneralGraphStats - Programm zum Speichern von Informationen über das EAN (wie zum Beispiel: größtes Intervall, kleinster Wert der Startlängen L_a auf einer Kante, etc.) in einer Datei
- FindCircles - Programm zum Finden von Zykeln
- MakeEAN - Programm mit graphischer Oberfläche zum manuellen Eingeben von Ereignis-Aktivitätsnetzwerken

Wir werden uns im Folgenden auf die Beschreibung der konkreten Implementierung der Heuristik beschränken. Da schon die Klasse Heuristik ohne Hilfsklassen ca. 1500 Zeilen lang ist und die komplette Vorstellung den Rahmen dieser Diplomarbeit sprengen würde, können nur wichtige Stellen vorgestellt werden. Die anderen implementierten Programme beschreiben wir kurz, werden jedoch nicht auf den Programmcode eingehen.
Der Ablauf der Heuristik ist in Abbildung 4.1 dargestellt.
Man könnte bei der Abfrage, ob die maximale Anzahl S der Iterationen erreicht ist, oder, ob der Fahrplan der aktuellen Iteration mit dem der letzten Iteration übereinstimmt, noch eine Abfrage hinzufügen, ob der aktuelle Fahrplan Π^i mit dem einer früheren Iteration Π^j mit $j < i - 1$ übereinstimmt. Denn dann gibt es einen Zykel und die Heuristik wird in einem lokalen Optimum hängenbleiben, bis die maximale Anzahl der Iterationen erreicht ist. Bricht man bei einem Zykel ab, so spart man Rechenzeit. Der Nachteil dieser Variante ist, dass man alle früheren Fahrpläne speichern muss und so ein höherer Speicherbedarf entsteht. Da die Speicherfreigabe bei Java nicht immer optimal ist, wurde bei der Implementation der Prozess des Findens von Zykeln ausgegliedert und in einem eigenständigen Programm gespeichert. Liefert die Heuristik einige Male hintereinander den selben Zielfunktionswert, so ist es ratsam, nach Zykeln zu suchen (s. Kapitel 4.2.4).
Im Gegensatz zu der Berechnung der M_e^2 ist es für die Heuristik nicht notwendig ein zusammenhängendes Netzwerk zu haben. Wir werden die Ergebnisse der Heuristik im unzusammenhängenden Netzwerk präsentieren.

4.2 Hilfsmittel

Die Implementierung der Heuristik arbeitet mit Instanzen, die von LinTim generiert wurden oder das LinTim Eingabeformat haben. Deshalb werden wir LinTim kurz vorstellen. Weiterhin gehen wir im Folgenden auf die anderen Hilfsmittel, die zum Implementieren der Heuristik beziehungsweise Erzeugen von zulässigen Instanzen benutzt werden, ein.

4.2.1 LinTim

LinTim [Lin11] ist ein Projekt der Universität Göttingen, das 2007 entstanden ist und zum Ziel hat Algorithmen zum Lösen von mathematischen Problemen in der Verkehrplanung zu finden und diese bereitzustellen. Es beinhaltet zur Zeit Algorithmen zur Linienplanung, zur periodischen Fahrplangestaltung, zum Verspätungsmanagement und zur Umlaufplanung sowie Datensätze. Die Datensätze sind im text/csv-artigen giv-Format gegeben und sehen zum Beispiel aus wie in Abbildung 4.2.
Um aus LinTim [Lin11] Instanzen für das aperiodische Fahrplanproblem mit OD-Paaren zu generieren, muss zunächst zu den bereits bestehenden Basisdateien (zu diesen gehört auch `OD.giv`), welche für jedes Netzwerk im Ordner `Basis` zu finden sind, mittels `make line-concept` ein Linienkonzept generiert und danach mit

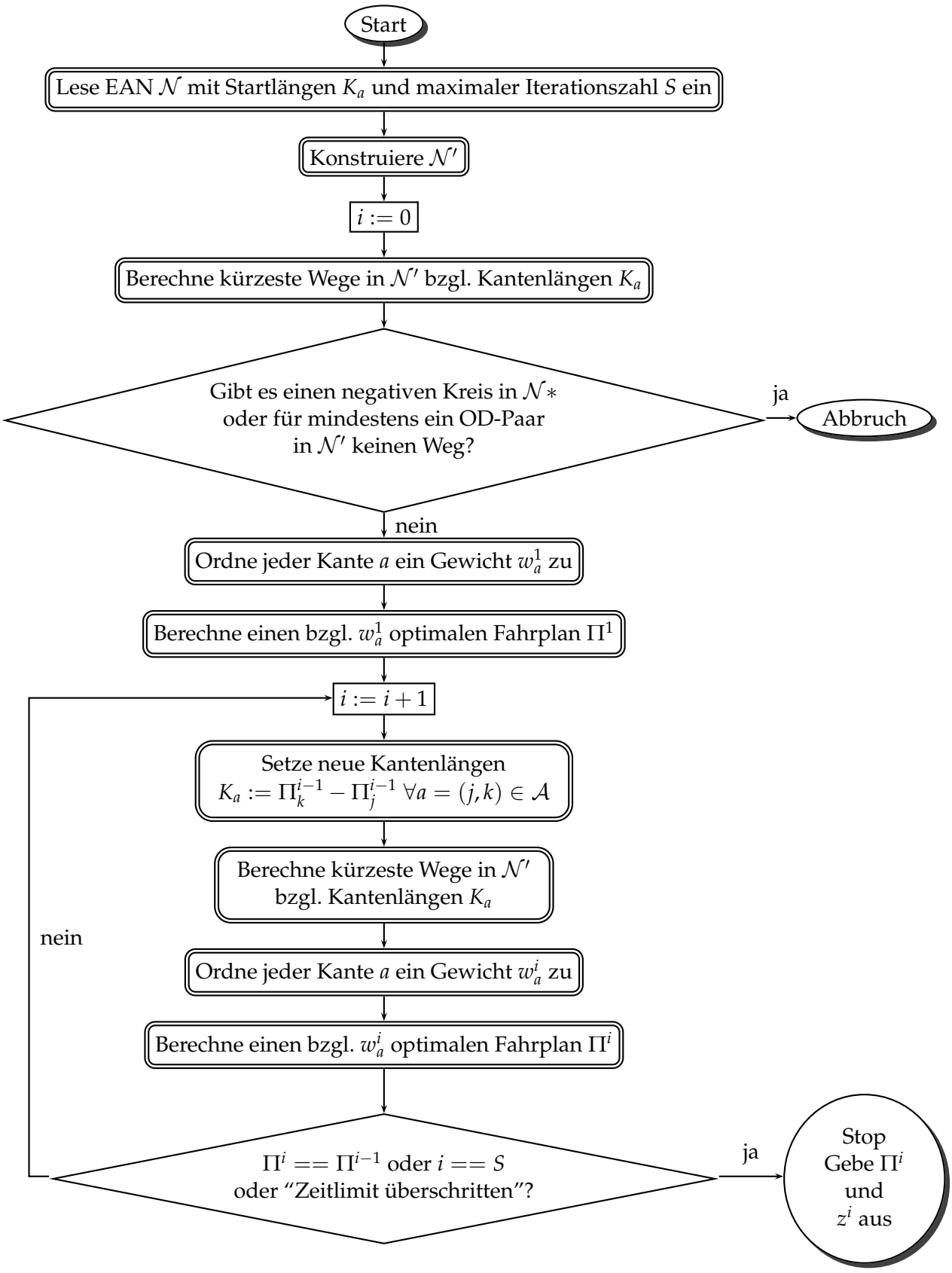


Abbildung 4.1: Ablauf der Heuristik

```
# small example of PTN
# traffic planning script, AS, 10.9.2007
# stop-id; short-name; long-name; x-coordinate; y-coordinate
1; 1; Eins; 100; 200
2; 2; Zwei; 200; 300
3; 3; Drei; 200; 200
4; 4; Vier; 200; 100
5; 5; Fuenf; 300; 100
6; 6; Sechs; 300; 200
7; 7; Sieben; 300; 300
8; 8; Acht; 400; 200
```

Abbildung 4.2: Beispiel eines PTNs ("Spiel") in LinTim

`make ptn2ean` ein periodisch zulässiges Ereignis-Aktivitätsnetzwerk erzeugt werden. Dies ist jedoch nicht unbedingt aperiodisch zulässig. Wir erhalten auf diese Weise die Datei `Events-periodic.giv`.
Wir berechnen mit Hilfe von `make periodic-timetable` einen periodischen Fahrplan und rollen diesen mit `make rollout` aus. Das bedeutet, wir vervielfachen alle periodischen Ereignisse und periodischen Aktivitäten mit Ausnahme der Headways entsprechend der Frequenz die in der Datei `Load.giv` angegeben ist. Dabei nehmen wir nur diese, die im angegebenen Ausrollzeitraum liegen. Wir erhalten auf diese Weise ein Ereignis-Aktivitätsnetzwerk ohne Headways (s. [Sch09]).

Headways sind Aktivitäten, die den Abstand zweier Abfahrtsereignisse bestimmen. Formal heißt das, es werden die Headway-Aktivitäten folgendermaßen definiert:

$$\mathcal{A}_{headway} = \{((v, l_1, dep), (v, l_2, dep)) \in \mathcal{E}_{dep} \times \mathcal{E}_{dep}\}$$

Dies macht insbesondere dann Sinn, wenn zwei Züge mit unterschiedlichen Linien l_1 und l_2 dasselbe Gleis am Bahnhof v benutzen und es deshalb einen Sicherheitsabstand zwischen ihnen geben soll.

Für das periodische Timetabling ist die Richtung der Headways egal. Beim aperiodischen Timetabling müssen wir die Reihenfolge der Ereignisse jedoch kennen. Ein Headway muss dabei immer eine Aktivität zwischen "früherem" Ereignis und späterem Ereignis sein. Um nun die Richtung der Headways festzulegen, rechnen wir den periodischen Fahrplan aus. Weiterhin wird für jede Headway-Kante eine umgedrehte Headway-Kante eingefügt, und auf Basis des berechneten Fahrplans entschieden, welche Kante für das aperiodisch zulässige Ereignis-Aktivitätsnetzwerk ausgewählt wird. Zu beachten ist, dass in der Datei `Global-Config.cnf` im Ordner `/LinTim/Examples` die Option `rollout_for_nonperiodic_timetabling` auf `true` gesetzt wird.
Um den Ausrollzeitraum zu verändern, können die Parameter `DM_earliest_time`

und DM_latest_time; verändert werden. Diese Parameter sind in Sekunden gegeben und als Referenz wird 0 : 00 genommen. Für die Default-Option [28800, 43200] bedeutet das also, dass wir im Zeitraum von 8 : 00 bis 12 : 00, also vier Stunden, ausrollen. Allerdings dürfen die Werte nicht "zu nah" beieinander liegen, da das resultierende Ereignis-Aktivitätsnetzwerk keine Ereignisse und Aktivitäten hätte und somit ein erfolgreiches Ausrollen nicht möglich wäre. Um Instanzen aus unterschiedlichen Linienkonzepten zu generieren, kann auch der Parameter lc_model verändert werden. Nach dem Ausrollen erhalten wir die Dateien Events-nonperiodic.giv und Activities-nonperiodic.giv und haben so unsere Instanz beisammen.

Die meisten Dateien besitzen einen Header, der die Datensätze beschreibt. Die Daten selbst sind tabellenartig aufgebaut, wobei die Semikolons die Spaltentrenner sind. Die Kommentare und Header werden durch # gekennzeichnet.

Wir stellen nun kurz die schon genannten Dateien, die zusammen als Eingabeinstanz für die Heuristik dienen, vor.

1. Events-periodic.giv liefert die periodischen Ereignisse mit eindeutiger ID (event_index), Art (Abfahrt- oder Ankunft, type = departure oder arrival), Haltestelle (stop), Linie (line) und die Anzahl an Passagieren (passengers).

```
# event_index; type; stop; line; passengers
1; "departure"; 1; 1; 20
2; "departure"; 1; 2; 0
3; "departure"; 1; 3; 50
4; "arrival"; 2; 1; 0
5; "arrival"; 2; 2; 0
6; "departure"; 2; 1; 50
```

   Wir werden nur die ID, die Art und die Haltestelle brauchen.

2. Events-nonperiodic.giv gibt die aperiodischen Ereignisse mit ID (event_id), periodischer ID (peridic_id), Typ (type), Zeit (time) und Anzahl an Passagieren (passengers) an.

```
# event-id; periodic-id; type; time; passengers
1; 1; "departure"; 1; 1
2; 2; "departure"; 1; 2
3; 3; "departure"; 1; 3
4; 4; "arrival"; 2; 1
5; 5; "arrival"; 2; 2
```

   Hier brauchen wir die eindeutige Event-ID, die periodische ID und den Typ. Es wird die periodische ID (peridic_id) mit der ID (event_index) aus Events-periodic.giv "gejoint" (vgl. [May05]). Konkret heißt das, wir machen einen Semi-Join über den event_index aus Events-periodic.giv und der periodic_id aus Events-nonperiodic.giv. Wir berechnen also zuerst das kartesische Produkt beider Tabellen und selektieren dann mit der Bedingung, dass

`event_index` und `periodic_id` identisch sind. Weiterhin projezieren wir auf die `event_id`, den `type` (aus `Events-nonperiodic.giv`) und den `stop` (aus `Events-periodic.giv`).

3. `Activities-nonperiodic.giv` liefert die Aktivitäten mit `activity-id`, periodischer ID (`periodic-id`), Typ (`type` = `drive`, `wait`, `change` oder `headway`), Startereignis (`tail-event`), Zielereignis (`head-event`), unterer Schranke L_a (`lower-bound`), oberer Schranke U_a (`upper-bound`) und der Anzahl der Passagieren (`passengers`).

```
# activity-id;periodic-id;type;tail-event-id;head-event-id;
# lower-bound;upper-bound;passengers
1; 1; "drive"; 1; 4; 1; 5; 3.0
2; 2; "drive"; 2; 5; 1; 2; 6.666666666666667
3; 3; "drive"; 3; 10; 9; 10; 3.0
4; 4; "wait"; 4; 6; 2; 8; 5.0
5; 5; "change"; 5; 6; 1; 1; 2.0
6; 6; "wait"; 5; 7; 3; 6; 1.0
```

   Wir brauchen die Activity-ID, den Typ, Start- und Zielereignis (werden gejoint mit events-nonperiodic ID), untere Schranke und obere Schranke.

4. `OD.giv` gibt die OD-Menge an:

```
# small example of PTN
# traffic planning script, AS, 10.9.2007
# left-stop-id; right-stop-id; customers
1;2;6
1;3;4
```

   Die Stops werden über die `stops` aus `Events-periodic.giv` und dann mit `perid` aus `Events-periodic.giv` gejoint. Die `customers` geben an wieviele Leute vom Stop `left-stop` nach Stop `right-stop` wollen.

4.2.2 Java

Da die Programmiersprache Java [Jav11] plattformunabhängig ist, (zur Zeit noch) unter der freien GNU General Public License [GNU11] steht und objektorientiert ist, wurde für die Implementierung Java verwendet. Klassen, die nicht zur Java-Klassenbibliothek gehören, werden im Folgenden explizit angegeben.

JGraphT

JGraphT ist eine Java-Bibliothek, die graphentheoretische Objekte und Algorithmen zur Verfügung stellt. [NC11] Für unsere Zwecke brauchen wir gerichtete, gewichtete

Graphen. Wir benutzen die Version 0.8.1. Die neuere Version 0.8.2 unterstützt keine FibonacciHeapNode-Objekte mehr, die für die Berechnung der kürzesten Wege gebraucht werden. Deshalb sollte explizit darauf geachtet werden Version 0.8.1 zu verwenden.

Gurobi

Für die Implementierung des Timetabling-Schrittes der Heuristik muss ein (LP-)Solver benutzt werden. Gurobi Optimization [Gur11] ist ein kommerzielles Softwarepaket, mit dem man gemischtganzzahlige lineare Programme lösen kann. Da Gurobi aber auch eine eingeschränkte freie Lizenz sowie eine uneingeschränkte akademische Lizenz anbietet und eine Java-Schnittstelle existiert, wird für das Timetabling Gurobi verwendet.

Linux

Alle selbstgeschriebenen Programme wurden für die Benutzung mit Linux entwickelt und getestet (genauer: Ubuntu 10.04 und Ubuntu 11.04). Da sich die Pfadangabe unter den Microsoft Windows Systemen (hier werden "\" benutzt) von der Pfadangabe unter Linux ("/") unterscheidet, ist es zur Zeit nicht möglich, die Programme unter Windows laufen zu lassen.

4.2.3 Erzeugen von guten OD-Paaren

Es wird sehr oft vorkommen, dass die Instanzen, die mittels LinTim erzeugt wurden, OD-Paare beinhalten, wo es für ein oder mehrere OD-Paare keinen Weg und somit insbesondere auch keinen kürzesten Weg gibt. Um dies zu vermeiden, ist es sinnvoll diese OD-Paare vor dem Aufruf der Heuristik auszusortieren. Dies geschieht mit dem Aufruf `java MakeGoodODs`. Das Programm speichert in einer neuen Datei mit randomisiertem Namen die OD-Paare $(u, v) \in OD$ aus der in der Konfigurationsdatei angegebenen OD-Datei, für die es einen Weg von u nach v gibt. Es wird dabei davon ausgegangen, dass die Ausgangs-OD-Datei symmetrisch ist. Das heißt, wenn es einen Weg von u nach v und von v nach u gibt, werden beide OD-Paare gespeichert.
Möchte man eine zufällige Auswahl einer festen oder zufälligen Anzahl an OD-Paaren aus der in der Konfigurationsdatei vorgegebenen OD-Datei auswählen, so wird der Aufruf `java ChooseODs [anz]` für anz OD-Paaren oder `java ChooseODs` für eine zufällige Anzahl getätigt.

4.2.4 Finden von Zykeln

Um Speicherplatz und Rechenzeit zu sparen, wurde das Überprüfen auf Zykel in ein eigenständiges Programm ausgelagert. Der Aufruf hierfür lautet

`java -jar FindCircles PfadZurFahrplandatei`. Läuft das Programm ohne eine Ausgabe durch, so wurde kein Zykel gefunden. Existiert ein Zykel, so werden die Iterationen des ersten und zweiten Auftauchens desselben Fahrplans ausgegeben. Auch der entsprechende Fahrplan wird angezeigt.

4.2.5 Kürzeste-Wege-Verfahren

Die Kürzesten-Wege-Verfahren wurden von Michael Siebert geschrieben und für unsere Zwecke entsprechend angepasst. Es gibt drei verschiedene Methoden:

- `Bellman-Ford` (`BF`)
- `TreeMapQueue` (`TMQ`)
- `FibonacciHeap` (`FH`)

Da Bellman-Ford bereits in Kapitel 2.2 beschrieben wurde, werden wir nun nur noch kurz `TreeMapQueue` und `FibonacciHeap` beschreiben.

TreeMapQueue

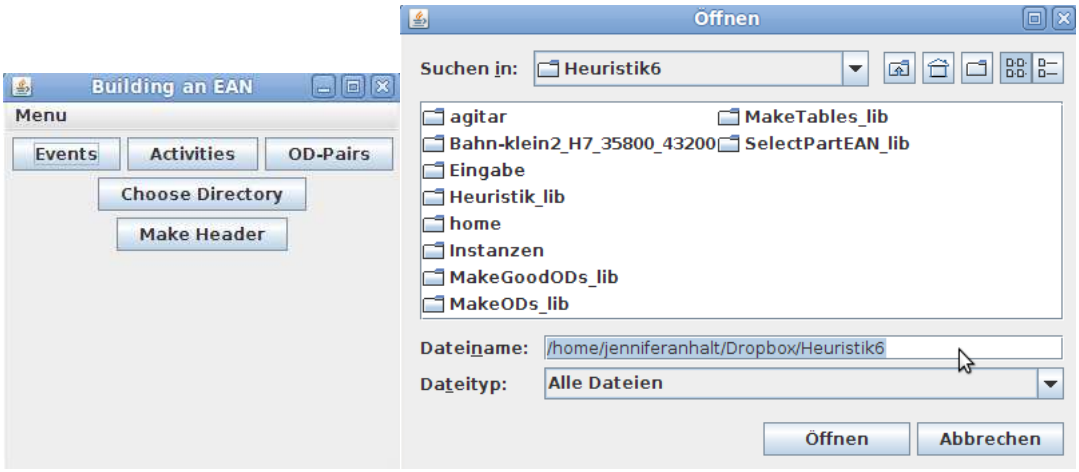
Die `TreeMapQueue`-Methode ist ein Dijkstra-Algorithmus (vgl. 2.2.13), der als Prioritätswarteschlange eine TreeMap benutzt.

FibonacciHeap

Auch die `FibonacciHeap`-Methode ist ein Dijkstra-Algorithmus. Hierbei wird als Prioritätswarteschlange jedoch ein FibonacciHeap verwendet. Die Laufzeit der Methode ist $\mathcal{O}(m + n \log n)$ (s. [CLRS07]).

4.2.6 Erzeugung von kleineren Instanzen

Da nicht alle Daten aus den LinTim-generierten Dateien gebraucht werden, kann es sehr mühsam sein, für das Erzeugen von eigenen kleinen Beispielen die entsprechenden Dateien zu schreiben. Deshalb ist es sinnvoll das Programm `MakeEAN` mit `java -jar MakeEAN.jar` aufzurufen. Es erscheint eine graphische Oberfläche (Abbildung 4.3a). Zunächst sollte das Verzeichnis, in dem die Instanz gespeichert werden soll, über die Schaltfläche `Choose Directory` ausgewählt werden (Abbildung 4.3b). Dann können die relevanten Daten für die Ereignisse (Abbildung 4.3), Aktivitäten (Abbildung 4.4) und OD-Paare (Abbildung 4.5) eingegeben beziehungsweise gelöscht werden. Zum Schluss kann mit dem Shortcut `Alt+1` der Autor der Instanz verändert werden. Mit `Make Header` werden Autor und Datum des Erstellens als Header in alle erzeugten Dateien geschrieben.



(a) Das Menü (b) Auswählen des Verzeichnisses

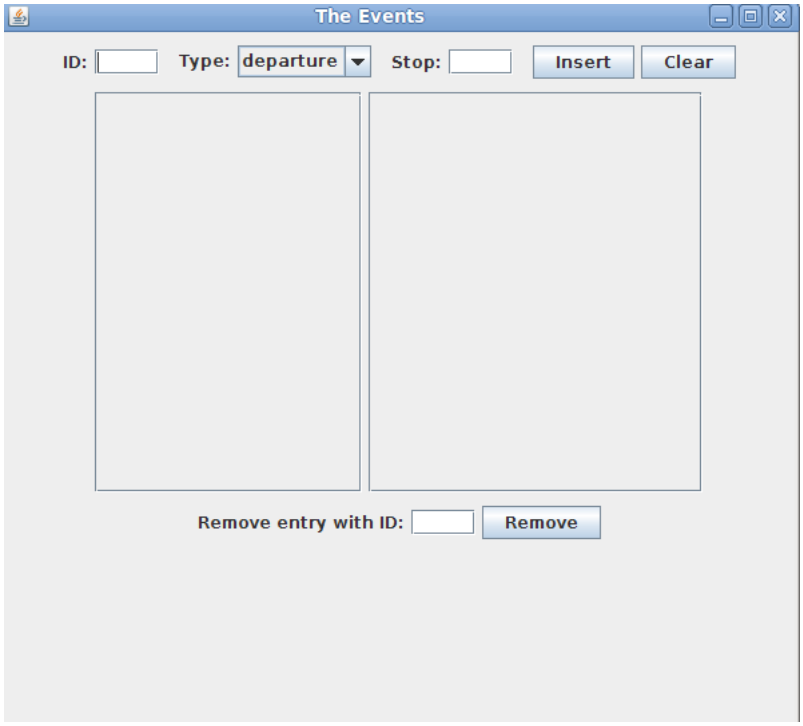


Abbildung 4.3: Eingabe der Ereignisse

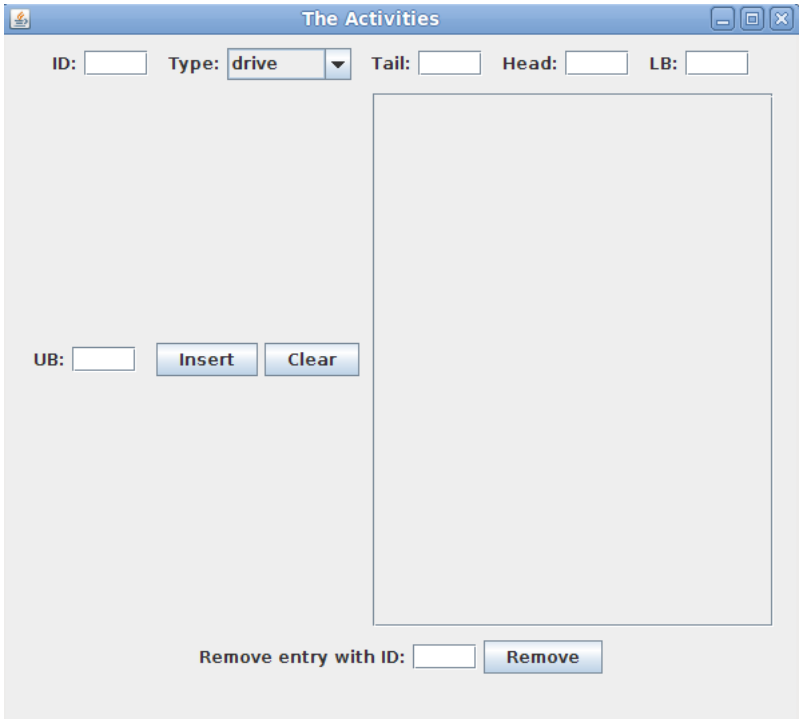


Abbildung 4.4: Eingabe der Aktivitäten

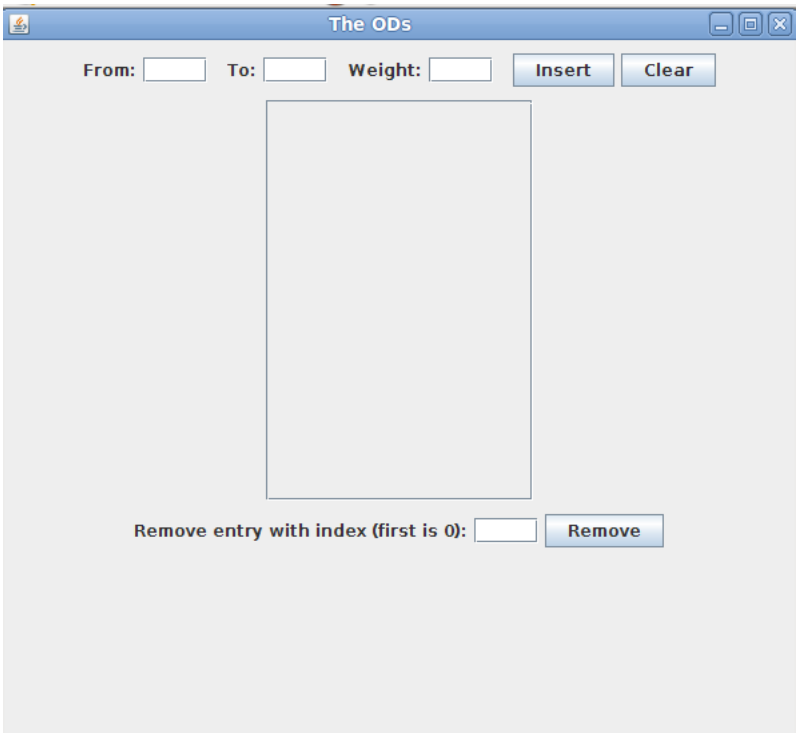


Abbildung 4.5: Eingabe der OD-Paare

4.3 Implementierung der Heuristik

4.3.1 Die Testklasse

Zum Aufruf der Heuristik muss eine Testklasse vorhanden sein, die ein `Heuristik`-Objekt erzeugt und anschließend eine der beiden Methoden

- `public void compute(String[] args){...}`

oder

- `public double computeoneodpair(String[] args, int line){...}`

aufruft (hier `compute`):

```
1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  public class HeuristikTest {
7    public static void main(String[] args){
8    Heuristik h = new Heuristik();
9    h.compute(args);
10   }
11 }
```

Dabei nimmt man die Methode `compute`, wenn man das aperiodische Fahrplanproblem mit OD-Menge OD lösen möchte, wobei $\#OD > 1$. Hat man das Ziel, das Problem für jedes OD-Paar $(u, v) \in OD$ unabhängig voneinander zu lösen, so ruft man iteriert (=$\#(OD)$-mal) die Methode `computeoneodpair` auf und übergibt der Methode zusätzlich noch die Zeilenanzahl `line` aus der OD-Datei, wo sich das entsprechende OD-Paar befindet. Da sich die beiden Methoden im Wesentlichen nur beim Einlesen der OD-Paare (beziehungsweise des einen OD-Paars) unterscheiden, beschränken wir uns hier auf das Präsentieren des Ablaufs beim Aufruf von `compute`.

4.3.2 Einlesen der Dateien und Aufbau des Graphen

Um die vier Dateien `Activities-nonperiodic.giv`, `Events-periodic.giv`, `Events-nonperiodic.giv` und `OD.giv` einzulesen, muss die Heuristik wissen, wo diese Dateien gespeichert sind (`default_inputfolder`, `default_outputfolder`, `default_nonperiodic_events`, `default_periodic_events`, `default_nonperiodic_activities`, `default_ods`).
Dies geschieht über die Datei `Config.cnf` in Abbildung 4.6.
Zu den weiteren Parametern, die angegeben werden müssen, gehören die Startlängen für die Längen der Kanten $K_a \forall a \in \mathcal{A}$ (`default_startlsg`). Die Option `LA` beziehungsweise `UA` gibt hierbei an, dass die unteren Schranken L_a beziehungsweise die oberen

```
default_inputfolder; Dropbox/Heuristik6/Instanzen/RelativerFehler_V2/
default_outputfolder; Dropbox/Heuristik6/Instanzen/RelativerFehler_V2/
default_nonperiodic_events; Events-nonperiodic.giv
default_periodic_events; Events-periodic.giv
default_nonperiodic_activities; Activities-nonperiodic.giv
default_ods; OD.giv
default_startlsg; LA
default_number_it; 100
method; TREE_MAP_QUEUE
write_paths_file; true
write_distribution_file; true
use_headways; false
# exact_method = virtarc or exact_method = edgebased
exact_method; edgebased
good_M; true
use_computed_M; true
use_LB; true
time_limit; 7200
```

Abbildung 4.6: Die Konfigurationsdatei

Schranken U_a der Intervalle $[L_a, U_a]$ $\forall a \in \mathcal{A}$ verwendet werden sollen. Ist `halb` eingestellt, so wird $\frac{L_a+U_a}{2}$ $\forall a \in \mathcal{A}$ verwendet. Bei der Option `r` werden zufällig generierte Längen benutzt.

Eingestellt wird außerdem die maximale Anzahl der Iterationen S `default_number_it` und die Kürzeste-Wege-Methode `method`. Hier gibt es die drei Optionen:
`TREE_MAP_QUEUE`, `FIBONACCI_HEAP` und `BELLMAN_FORD`.

Mit der Option `write_paths` wird eingestellt, ob eine Datei erstellt wird, die für jede Iteration und jedes OD-Paar das erste Ankunftsereignis und das letzte Abfahrtsereignis speichert. Ist diese Option auf `true` gesetzt, muss man jedoch mit einer viel längeren Laufzeit rechnen.

Außerdem gibt es mit `write_distribution_file; true` die Möglichkeit die $RZ^{\Pi^k}_{w_a^j}$ für $k, j \in \mathbb{N}$ mit $j = k - 1$ in eine Datei zu schreiben.

Mit `use_headways; true` kann die Heuristik auch Headways verarbeiten. Hier soll jedoch gewarnt werden, dass zumindest einige Instanzen, die mit LinTim generiert wurden, mit Verwendung der Headways nicht mehr zulässig waren. Aus diesem Grund werden wir nur die Implementierung ohne Verwendung der Headways besprechen.

Die Punkte `exact_method` und `good_M` sind Einstellungsmöglichkeiten für $(ILP4)/(ILP5)$ (`edgebased`) und $(ILP6)/(ILP8)$ (`virtarc`) mit M_e^1 beziehungsweise M_v^1 (`good_M; true`) und M_e^2 beziehungsweise M_v^2 (`good_M; false`). Falls schon M_e^2

oder M_v^2 berechnet sind, und eine entsprechende Datei mit den M's vorhanden ist, kann diese über die Option `use_computed_M; true` genutzt werden. Damit kann unter Umständen die Rechenzeit sehr stark verkürzt werden. Wenn `use_LB; true` gesetzt wird, werden $(ILP5)$ und $(ILP8)$ mit unterer Schranke LB_2, falls sie in einer Statistik-Datei für die Instanz vorhanden ist, beziehungsweise LB_1, falls dies nicht der Fall ist, berechnet. Ansonsten werden $(IL4)$ beziehungsweise $(ILP6)$ berechnet. Mit der Option `time_limit` wird ein Zeitlimit für das Berechnen (nicht das Aufstellen oder das Berechnen der M's) des ganzzahligen linearen Programms gesetzt. Falls es kein Zeitlimit geben soll, wird `time_limit` auf 0 gesetzt.
Wenn keine Config.cnf-Datei vorhanden ist, werden Default-Parameter gesetzt und es wird versucht mit diesen weiterzuarbeiten.

Das Einlesen des Graphen geschieht in den Methoden

```
public void readGraphPerEvents() {...}

public void readGraphNonPerEvents() {...}

public void readGraphActivities () {...}

public int readODPairs (){...}
```

Diese Methoden lesen die vier Dateien, die in `default_periodic_events`, `default_periodic_events`, `default_nonperiodic_activities` und `default_ods` angegeben wurden, ein und füllen ArrayLists mit entsprechenden Objekten. Diese sind:

1. `PerEvent(int id, String type, int stop, int line)`
2. `NonPerEvent(int id, int perid, String type)`

   NonPerEvent hat eine weitere Klassenvariable `int stop`, die bei der Initialisierung auf den Defaultwert 0 gesetzt wird und erst später (in `make Stops()`) seinen richtigen Wert bekommt.
3. ```
   Activity(int id, String type,
            int tailevent, int headevent, int LB, int UB)
   ```
4. `ODPair(int leftstop, int rightstop, int passengers)`

Alle diese ArrayLists sind als Klassenvariablen implementiert.
In

```
makeStops(){...}
```

wird für jedes NonPerEvent die bisher mit 0 initialisierte Variable `int stops` gesetzt, wobei die Informationen aus `int perid` und `int stop` aus `PerEvent` geholt werden. Nach dem Aufruf dieser Methode ist die ArrayList `nonPeriodicEvents` nun mit allen für den weiteren Verlauf wichtigen Informationen ausgestattet.
In der folgenden Methode werden die Origin- und Destination-Knoten erzeugt:
```

```
public static int readVirtNodesArray() {...}
```

Gespeichert werden die Origin- und Destination-Knoten in einer ArrayList von `VirtNode`-Objekten, welche folgende Struktur haben:

```
VirtNode(int id, String type, int stop)
```

Weiterhin wird ein ShortestPathGraph-Objekt

```
 ShortestPathsGraph<Integer, Integer> sp =
new ShortestPathsGraph<Integer, Integer>();
```

erzeugt, das in den Methoden

```
public static void makeGraphEvents(
ShortestPathsGraph<Integer, Integer> v) {
 ...
}

public static void makeVirtNodes(
ShortestPathsGraph<Integer, Integer> v,
final int anzNonPerEv
) {
...
}
public static void makeActivities(
ShortestPathsGraph<Integer, Integer> sp,
int anzvirtnodes, ArrayList<StartLsg> start
) {
...
}
```

befüllt wird. Die Zahl `anzvirtnodes` ist dabei die Anzahl der Origin- und Destination-Knoten. Die ArrayList `start` enthält für alle Kanten die Länge, die die Startlänge ist. Im Fall von randomisierten Startlängen werden diese sichtbar für den Benutzer ausgegeben.

4.3.3 Berechnung der kürzesten Wege

Das Berechnen der kürzesten Wege geschieht in der Methode `computeShortPath`.

```
public static int computeShortPath(
  ShortestPathsGraph<Integer, Integer> sp,
  ArrayList<ShortPathOrd> shPathOrd, int anz) {
...
}
```

Neben dem ShortestPathsGraph sp wird der Methode eine ArrayList mit ShortPathOrd-Objekten `ShortPathOrd(int id, int gewicht)` übergeben, in denen die Ereignisse und Gewichte für die Zielfunktion gespeichert werden. Der Parameter `anz` gibt die aktuelle Iteration an.

Zunächst werden die Origin- und Destination-Knoten zusammen mit ihren Stops nach Arrival und Departure-Knoten sortiert und in zwei HashMaps
`HashMap<Integer, VirtNode> hmstopsArr = newHashMap<Integer,VirtNode>();`
und
`HashMap<Integer, VirtNode> hmstopsDep = new HashMap<Integer, VirtNode>();`
gespeichert.

Die eigentliche Berechnung der kürzesten Wege erfolgt hier:

```
int lastID = 0;
double sum = 0;
for (ODPair odPair : odPairs) {
  double op1 = 0;
  if (hmstopsDep.containsKey(odPair.getLeftstop())
     && hmstopsArr.containsKey(odPair.getRightstop())){
    VirtNode vnode1 = hmstopsDep.get(odPair.getLeftstop());
    VirtNode vnode2 = hmstopsArr.get(odPair.getRightstop());
    int i = vnode1.getId();
    try {
      if (lastID != i){
        sp.compute(i);
        lastID = i;
      }
      int j = vnode2.getId();
      LinkedList<Integer> pathpath = sp.trackPath(j);
      double lengthpath = 0;
      if (pathpath.size() != 0) {
        ShortPath a = new ShortPath(sp.getEdgeTarget(
                  pathpath.getFirst()
            ),
                                    sp.getEdgeSource(
             pathpath.getLast()
                  ),
                                    odPair.getPassengers());
        shPath.add(a);
        op1 = op1*odPair.getPassengers();
        if (paths==true && oneodpair==false){
          try {
            FileWriter pw = new FileWriter(pathfile, true);
            pw.write(odPair.getLeftstop() + "; " +
                     odPair.getRightstop() + "; " +
                     a.getStart() + "; " +
                     a.getZiel() + "; " +
```

```
35                      a.getGewicht() + "; " +
36                      lengthpath + "; " +
37                      lengthpath*a.getGewicht() + "\n");
38            pw.flush();
39            pw.close();
40          }
41          catch (IOException ioe) {
42            System.out.println(ioe);
43          }
44        }
45      }
46      else {
47        System.out.println("Es gibt keinen Weg fuer OD-Paar (" +
48                           odPair.getLeftstop() +
49                           ", " +
50                           odPair.getRightstop() + ")
51                von " + i + " nach " + j);
52        return 1;
53      }
54    }
55    catch (GraphMalformedException e) {
56      e.printStackTrace();
57    }
58  }
59  else {
60    System.out.println(odPair.getLeftstop() +
61                       " -> " + odPair.getRightstop() +
62                       " ist nicht in den OD-Paaren vorhanden.");
63  }
64  sum = sum + op1;
65 }
```

Für jedes OD-Paar wird der Origin-Knoten `i` aus `hmstopsDep` und der Destination-Knoten `j` aus `hmstopsArr` geholt und von `i` ausgehend werden mittels `compute(i)` kürzeste Wege zu allen Knoten $\neq$ `i` berechnet, falls der Leftstop vom aktuellen OD-Paar in `hmstopsDep` und der Rightstop in `hmstopsArr` vorhanden ist. Mithilfe von `sp.trackPath(j);` wird dann ein kürzester Weg von `i` nach `j` bestimmt. Da dies für alle OD-Paare geschieht, sparen wir uns ein paar Aufrufe von `sp.compute(i)`, wenn die OD-Paare geordnet nach ihrem Leftstop gespeichert sind, denn es wird in den Zeilen 11-14 abgefragt, ob das vorhergehende Berechnen der kürzesten Wege von denselben Origin-Knoten geschehen ist. Wenn dies der Fall ist, wird `compute` nicht aufgerufen.

Wenn das Aufrufen von `trackPath` keine leere Liste geliefert hat, gibt es einen Weg von *i* nach *j*.

```
66 ...
67 System.out.println("Sorting shortest Paths...");
```

```
68 HashMap<Integer, Integer> hmShPathOrd=
69 new HashMap<Integer, Integer>();
70 hmShPathOrd.clear();
71 int shpathsize=shPath.size();
72 int nonperevsize=nonPeriodicEvents.size();
73 for (int i = 0; i < shpathsize; i++) {
74 //Wenn der Startknoten noch nicht vorhanden ist,
75 //fuege ihn mit negativem Gewicht ein
76   if (!hmShPathOrd.containsKey(shPath.get(i).getStart())) {
77     hmShPathOrd.put(shPath.get(i).getStart(),
78                     -shPath.get(i).getGewicht());
79   }
80   else {
81     //Wenn der Startknoten vorhanden ist, speichere Wert,
82     //entferne Eintrag aus HashMap und fuege mit Gewicht
83     //des alten minus das des Knotens hinzu
84     int temphmvar;
85     temphmvar=hmShPathOrd.get(shPath.get(i).getStart()).intValue();
86     hmShPathOrd.remove(shPath.get(i).getStart());
87     hmShPathOrd.put(shPath.get(i).getStart(),
88                     -shPath.get(i).getGewicht() + temphmvar);
89   }
90    //Wenn der Zielknoten noch nicht vorhanden ist,
91    //fuege ihn mit negativem Gewicht ein
92   if (!hmShPathOrd.containsKey(shPath.get(i).getZiel())) {
93     hmShPathOrd.put(shPath.get(i).getZiel(),
94                     shPath.get(i).getGewicht());
95   }
96   else {
97     //Wenn der Zielknoten vorhanden ist,
98     //speichere Wert, entferne Eintrag aus HashMap
99     //und fuege mit Gewicht des alten plus das des Knotens hinzu
100    int temphmvar;
101    temphmvar=hmShPathOrd.get(shPath.get(i).getZiel()).intValue();
102    hmShPathOrd.remove(shPath.get(i).getZiel());
103    hmShPathOrd.put(shPath.get(i).getZiel(),
104                    shPath.get(i).getGewicht() + temphmvar);
105  }
106 }
```

Wenn ein Weg $P = (i, v_1, \ldots, v_n, j)$ für OD-Paar (i, j, w_{ij}) vorhanden ist, wird das ShortPath-Objekt, bestehend aus Ereignis v_1 und v_n mit Gewicht w_{ij} in die ArrayList eingefügt. Da Gurobi keine faktorisierten Ausdrücke kennt, wird $w_{ij} \cdot (\Pi_n - \Pi_1)$ für alle Wege $P = (i, v_1, \ldots, v_n, j)$ noch in die Form $w_{ij} \cdot (\Pi_n - \Pi_1) = w_{ij} \cdot \Pi_n - w_{ij} \cdot \Pi_1$ gebracht (Zeilen 66-106) und als ShortPathOrd-Objekt in die ArrayList `shPathOrd` gespeichert (Zeilen 107-120).

```
107 for (int j = 0; j < nonperevsize; j++) {
108   if (hmShPathOrd.containsKey(nonPeriodicEvents.get(j).getId())) {
109     ShortPathOrd temp=new ShortPathOrd(
110         nonPeriodicEvents.get(j).getId(),
111         hmShPathOrd.get(nonPeriodicEvents.get(j).getId()).intValue()
112         );
113     shPathOrd.add(temp);
114   }
115   else {
116     ShortPathOrd temp=
117          new ShortPathOrd(nonPeriodicEvents.get(j).getId(), 0);
118     shPathOrd.add(temp);
119   }
120 }
```

Die Zeilen 30 - 39 schreiben die berechneten Wege für jedes OD-Paar in der Form `#Leftstop;Rightstop;firstnode;lastnode;pass;length;length*pass` in eine Datei, falls `write_paths_file` auf true gesetzt ist und nicht die Methode `computeoneodpair` benutzt wird. Dabei ist `pass` die Anzahl der Passagiere w_{uv} mit Leftstop u und Rightstop v. Das heißt, in diesem Fall wird für jedes OD-Paar der erste Departure-, sowie der letzte Arrivalknoten (`firstnode` und `lastnode`), die Passagierzahlen (`pass`), die Länge des Weges `length` und die Länge des Weges multipliziert mit der Anzahl der Passagiere (`length*pass`) geschrieben.

4.3.4 Der Timetabling-Schritt

Um einen optimalen Fahrplan zu berechnen, wird die Methode

```
public static boolean makeOptTimetable(
ArrayList<ShortPathOrd> shPathOrd,
HashMap<Integer, Integer> timtab,
int anzloop
) {
...
}
```

aufgerufen. In `HashMap<Integer, Integer> timtab` wird der berechnete Fahrplan gespeichert. `shPathOrd` ist die ArrayList, in der die zuvor berechneten kürzesten Wege zu finden sind. `anzloop` ist die aktuelle Iteration.

Die Zielfunktion und gleichzeitig auch die Π_i werden in folgender Weise erstellt:

```
1 GRBVar[] nbVars = new GRBVar[nonPeriodicEvents.size()];
2 int nonperevsize = nonPeriodicEvents.size();
3 int shpathordsize = shPathOrd.size();
4 for (int i = 0; i < nonperevsize; i++) {
5   for (int j = 0; j < shpathordsize; j++) {
```

```
6       if (nonPeriodicEvents.get(i).getId()
7           ==
8     shPathOrd.get(j).getId()) {
9         nbVars[i] = model.addVar(0,
10          GRB.INFINITY,
11          shPathOrd.get(j).getGewicht(),
12          GRB.CONTINUOUS,
13          String.valueOf(
14            shPathOrd.get(j).getId())
15          );
16      }
17    }
18    if (nbVars[i] == null) {
19     nbVars[i] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
20      String.valueOf(nonPeriodicEvents.get(i).getId()));
21    }
22  }
```

Ein Array von GRBVar wird erzeugt, das als Größe die Anzahl der Elemente von `nonPeriodicEvents`, also die Anzahl der Ereignisse im Netzwerk, hat. Weiterhin wird überprüft, ob das Ereignis mit einem aus der Berechnung der kürzesten Wege übereinstimmt. Falls dies der Fall ist, so wird dieses mit dem entsprechenden Gewicht zur Summe in der Zielfunktion hinzugefügt. Im anderen Fall geht es mit dem Gewicht 0 in die Summe ein.

Durch `model.set(GRB.IntAttr.ModelSense, 1)` wird die Minimierung der Zielfunktion eingestellt.

Nun fehlen noch die Nebenbedingungen $\Pi_j - \Pi_i \in [L_a, U_a]\ \forall a = (i,j) \in \mathcal{A}$:

```
23  ...
24  //Nebenbedingungen
25  int activitiessize = activities.size();
26  for (int i = 0; i < activitiessize; i++) {
27    if (useheadways == false){
28      if (!activities.get(i).getType().equals("\"headway\"")) {
29        GRBLinExpr expr = new GRBLinExpr();
30        expr.addTerm(-1.0, nbVars[activities.get(i).getTailevent() -
              1]);
31        expr.addTerm(1.0, nbVars[activities.get(i).getHeadevent() -
              1]);
32        //Um $\Pi_j-\Pi_i\leq UB und \geq LB$ zu erreichen im expr ein
            - spendiert, da  bei addTerm addiert wird
33        model.addConstr(expr, GRB.LESS_EQUAL, activities.get(i).getUB
              (), "u" + i);
34        model.addConstr(expr, GRB.GREATER_EQUAL, activities.get(i).
              getLB(), "l" + i);
35      }
```

```
36    }
37    else{
38      GRBLinExpr expr = new GRBLinExpr();
39      expr.addTerm(-1.0, nbVars[activities.get(i).getTailevent() - 1])
          ;
40      expr.addTerm(1.0, nbVars[activities.get(i).getHeadevent() - 1]);
41      //Um $\Pi_j-\Pi_i\leq UB und \geq LB$ zu erreichen im expr ein -
          spendiert, da bei addTerm addiert wird
42      model.addConstr(expr, GRB.LESS_EQUAL, activities.get(i).getUB(),
          "u" + i);
43      model.addConstr(expr, GRB.GREATER_EQUAL, activities.get(i).getLB
          (), "l" + i);
44    }
45  }
```

Wenn `useheadways == false` gesetzt ist, werden die Headways nicht mit berücksichtigt. Das `GRBLinExpr expr`-Objekt ist ein linearer Ausdruck, der durch Addieren der einzelnen Terme entsteht.
Durch `model.addConstr(expr,GRB.LESS_EQUAL,activities.get(i).getUB(),"u"+i)` wird der lineare Ausdruck `expr` zu einer Nebenbedingung mit Relationszeichen `GRB.LESS_EQUAL` := $\leq$ beziehungsweise `GRB.GREATER_EQUAL` := $\geq$ als zweitem Argument und der rechten Seite als drittem Argument hinzugefügt.

`model.optimize()` versucht das Problem zu lösen. Falls das Problem nicht unbeschränkt ist und eine Optimallösung besitzt, so werden die entsprechenden $\Pi_i \; \forall i \in \mathcal{E}$ in das Array `nbVars` geschrieben und es kann über `nbVars[i].get(GRB.DoubleAttr.X)` auf den Wert zugegriffen werden.

Anschließend werden noch der Zielfunktionswert und der Fahrplan in die entsprechenden Ausgabedateien geschrieben.

4.3.5 Das Setzen der neuen Längen

Nach jedem Timetabling-Schritt werden den Kanten, wenn die Abbruchbedingung nicht erfüllt ist, neue Längen gegeben, auf deren Basis ein weiteres Mal die kürzesten Wege berechnet werden. Das Setzen dieser Längen geschieht in der Methode:

```
public static void setNewLengths(
ShortestPathsGraph<Integer, Integer> sp,
HashMap<Integer, Integer> timtab) {
...
}
```

Da die Längen aus dem Fahrpan Π^k, im zuvor getätigten Timetabling-Schritt berechnet, als $\Pi^k_j - \Pi^k_i \; \forall (i,j) \in \mathcal{A}$ gesetzt werden, muss der entsprechende Fahrplan `timtab` übergeben werden.

4.3.6 Abbruchbedingung

Abgebrochen wird die Heuristik, nachdem eine bestimmte Anzahl an maximalen Iterationen erreicht ist, die Fahrpläne in zwei aufeinanderfolgenden Iterationen übereinstimmen oder ein vorgegebenes Zeitlimit (in unserem Fall 4 Stunden) erreicht ist.

Für die Iteration bis zur maximalen Anzahl S wird eine `do-while`-Schleife benutzt, wobei überprüft wird, ob die aktuelle Anzahl an Iterationen $\leq S$ ist und der Fahrplan Π^k der aktuellen Iteration k nicht mit dem Fahrplan Π^{k-1} aus der vorhergehenden Iteration $k-1$ übereinstimmt. Wenn mindestens eine der beiden Bedingungen nicht erfüllt ist, wird die Heuristik abgebrochen und der Fahrplan Π^k ausgegeben. Weiterhin wird überprüft, ob die Zeitspanne überschritten ist, und auch in diesem Fall wird die Heuristik abgebrochen und der letzte berechnete Fahrplan ausgegeben.

5 Ergebnisse

Um die Performance sowie die Genauigkeit der Heuristik in der Praxis zu untersuchen werden sie mit LinTim-Instanzen (s. Kapitel 4.2.1) als Eingabe ausgeführt. Die Instanzen basieren auf den öffentlichen Verkehrsnetzwerken Bahn-klein, Bahn-gross, Spiel und Athens-Metro. Aus diesen wurden mit unterschiedlichen Modellen Linienkonzepte erstellt, wodurch zulässige Instanzen für das periodische Fahrplanproblem mit OD-Paaren gewonnen werden konnten. Ein periodischer Fahrplan wurde berechnet und schließlich wurde dieser mit verschiedenen Zeiträumen ausgerollt, so dass ein aperiodisch zulässiger Fahrplan (Ereignis-Aktivitätsnetzwerk $\mathcal{N}$ und OD-Paare, es gibt für jedes OD-Paar einen Weg und es gibt keinen Kreis negativer Länge in $\mathcal{N}'$) generiert wurde.

5.1 Die Hardware und Konfiguration

Während der Untersuchungen wurde die Hardware umgestellt, weswegen es nötig war auf zwei verschiedene Rechnertypen zuzugreifen.

- 64-Bit-Rechner mit 12GB Arbeitsspeicher und 3 Prozessoren vom Typ "Dual Core AMD Opteron(tm) Processor 275" mit einer Taktfrequenz von jeweils 2200 MHz
- 64-Bit-Rechner mit 9.8GB Arbeitsspeicher und 1 Prozessor vom Typ "SunFire X4200, AMD Opteron 2.2 GHz"

Die Tests für die Heuristik wurden mit der folgenden Konfiguration gemacht:

- `write_paths_file; false`
- `write_distribution_file; true`
- `use_headways; false`

Als Startlängen wurden `LA`, `UA` und `halb` und als Kürzeste-Wege-Algorithmen `TreeMapQueue` und `FibonacciHeap` gewählt. Im Folgenden wird für die Startlängen `LA` mit Kürzeste-Wege-Algorithmus `TreeMapQueue` kurz `LA TMQ` geschrieben und für die Startlängen `LA` mit Kürzeste-Wege-Algorithmus `FibonacciHeap` kurz `LA FH`. Analog mit den anderen Kombinationen.
Für die Heuristik haben wir folgende Gurobi-Parameter genommen:

- Method = 4
- OutputFlag = 0

Für die ganzzahligen linearen Programme wurden folgende Gurobi-Parameter gesetzt

- Method = 4
- MIPGap = 0.000000001
- FeasibilityTol = 0.000000001
- Heuristics = 0.05 (Default) und Heuristics = 0

und folgende Konfigurationseinstellungen vorgenommen:

- `exact_method; virtarc` oder `exact_method; edgebased`
- `use_computed_M; true` (da, wo M's schon vorhanden waren)
- `use_LB; false` oder `use_LB; true`
- `time_limit; 7200`

Dabei bedeutet Method = 4, dass ein deterministischer Solver gewählt wird. Allerdings wird dies durch das Zeitlimit von 7200 Sekunden = 2 Stunden aufgehoben. Es können also bei mehreren Durchläufen, deren Zeitlimit ausgereizt ist, durch Belastung des Servers oder andere äußere Umstände unterschiedliche Zielfunktionswerte beobachtet werden. OutputFlag = 0 deaktiviert die Zwischenausgabe des Gurobi-Solvers. Das heißt, es wird nicht jede Änderung im Branch & Cut oder Simplex-Algorithmus angezeigt. Der Parameter MIPGap gibt die relative Optimalitätslücke an, bei der der Solver abbricht. FeasibilityTol gibt die primale Zulässigkeitstoleranz an. Das heißt, alle Nebenbedingungen des primalen Programms müssen bis auf den Faktor FeasibilityTol erfüllt sein. Mit Heuristics wird (in Prozent) angegeben, wieviel Wert der Solver auf Heuristiken legt.

5.2 Die Eingabeinstanzen der Heuristik

In Tabelle 5.1 sind die Instanzen aufgelistet, mit denen Algorithmus 3.4.1 getestet wurde. Man kann am Namen der Instanz erkennen, auf welchem Netzwerk sie basiert (Bahn-gross, Bahn-klein, Athens-Metro, Spiel), wie das Linienkonzept erstellt wurde (H7, Xpress) und wie der Ausrollzeitraum ist. Bahn-klein2_H7_28800_43200 basiert also auf dem Bahn-klein-Netzwerk. Die 2 gibt an, dass die Frequenzen im Linienpool des Netzwerks verändert (weniger) wurden, dass das Linienkonzept H7 ist und dass der Ausrollzeitraum (bestimmt durch die Parameter `DM_earliest_time`

Instanz	Abkürzung
Bahn-gross2_H7_35800_43200	BG2H7_1
Bahn-gross_Xpress_28800_43200	BGX_1
Bahn-gross_Xpress_35800_43200	BGX_2
Bahn-gross_Xpress_36800_38000	BGX_3
Bahn-gross3_Xpress_41800_43200	BG3X_1
Bahn-klein2_Xpress_28800_43200	BK2X_1
Bahn-klein2_Xpress_35800_43200	BK2X_2
Bahn-klein2_H7_28800_43200	BK2H7_1
Bahn-klein2_H7_35800_43200	BK2H7_2
Spiel_H7_35800_43200	SH7_1
Athens-Metro_Xpress_21800_43200	AMX_1
Athens-Metro_Xpress_28800_43200	AMX_2
Athens-Metro_Xpress_35800_43200	AMX_3

Tabelle 5.1: Die Instanzen und ihre Abkürzungen

und
`DM_latest_time`) 28800 bis 43200 beträgt.

Die Parameter der getesteten Instanzen sind in den Tabellen 5.2, 5.3 und 5.4 abzulesen.

5.3 Die Ergebnisse

Während des Testens der Instanzen musste von Gurobi 3.0.0 auf Gurobi 4.5.1 gewechselt werden. Der Grund hierfür war eine auslaufende Lizenz. Leider war zu diesem Zeitpunkt keine Lizenz mehr für Gurobi 3.0.0 zu erhalten. In Tabelle A.2 ist eingetragen, welcher Rechner und welche Gurobiversion für die jeweiligen Ergebnisse verwendet wurde.

Die wesentlichen Ergebnisse unter Gurobi 3.0.0 sind in Tabelle A.3 zusammengefasst. Diejenigen für Gurobi 4.5.1 sind in Tabelle 5.5 zu finden.

Es fiel auf, dass die Ergebnisse teilweise nicht mehr mit denen vor dem Upgrade übereinstimmten. Der Wert der Zielfunktionswerte in der ersten Iteration stimmte jedoch überein. Dies ist damit zu erklären, dass Gurobi 4.5.1 intern einen anderen Algorithmus zur Bestimmung beziehungsweise Auswahl eines optimalen Fahrplans hat. In der ersten Iteration gibt es also mehrere Kandidaten für einen optimalen Fahrplan. Gurobi 4.5.1 wählt einen anderen Fahrplan als Gurobi 3.0.0. Beim Vergleich der alten und neuen Ergebnisse bestätigte sich diese Vermutung. Durch die unterschiedliche Wahl werden in der zweiten Iteration unterschiedliche kürzeste Wege gewählt und

Instanz	BG2H7_1	BGX_1	BGX_2	BGX_3	BG3X_1
# Events	11146	21466	10991	1706	1471
# Dep	5593	10757	5521	870	745
# Arr	5593	10709	5470	836	726
# Activities	19734	42419	19613	1297	1169
# Drive	4560	9736	4500	240	244
# Wait	5135	9904	5037	749	655
# Change	10039	22779	10076	308	270
# OD-Paare	6478	21682	3197	29	213
LB_2	4.40734848E9	2.039111976E10	4.65561948E9	155280.0	359220.0
UB	4.429344E9	2.09801511E10	4.68182442E9	155280.0	359220.0
# virt. Knoten	634	636	602	51	282
# Origins	317	318	301	24	138
# Destinations	317	318	301	27	144
kleinstes L_a	60	60	60	60	60
größtes L_a	6960	9120	6180	1080	1380
kleinstes U_a	60	60	60	60	60
größtes U_a	8100	10620	6600	3720	3720
kleinstes Intervall	[60,60]	[1560,1560]	[1560,1560]	[540,540]	[120,120]
kleinste Länge Intervall	0	0	0	0	0
größtes Intervall	[180,3720]	[180,3720]	[180,3720]	[180,3720]	[180,3720]
größte Länge Intervall	3540	3540	3540	3540	3540
# Zsghskomp.	202	142	188	502	398
# Isolierte Knoten	42	38	40	43	23

Tabelle 5.2: Kennziffern der Instanzen

Instanz	BK2X_1	BK2X_2	BK2H7_1	BK2H7_2
# Events	18542	9520	21092	10818
# Dep	9298	4780	10575	5428
# Arr	9244	4740	10517	5390
# Activities	32722	15416	36339	17074
# Drive	8492	3974	9674	4527
# Wait	8631	4408	9814	5003
# Change	15599	7034	16851	7544
# OD-Paare	15809	2770	15690	2709
LB_2	1.684126938E10	4.43634396E9	1.679175708E10	4.40392524E9
UB	1.721578188E10	4.46736618E9	1.711741872E10	4.42468074E9
# virt. Knoten	497	484	496	484
# Origins	248	244	248	242
# Destinations	249	240	248	242
kleinstes L_a	60	60	60	60
größtes L_a	9120	6180	9120	7020
kleinstes U_a	60	60	60	60
größtes U_a	9420	6840	9420	7920
kleinstes Intervall	[1680,1680]	[1680,1680]	[1680,1680]	[1680,1680]
kleinste Länge Intervall	0	0	0	0
größtes Intervall	[180,3720]	[180,3720]	[180,3720]	[180,3720]
größte Länge Intervall	3540	3540	3540	3540
# Zsghskomp.	96	139	144	182
# Isolierte Knoten	33	34	33	34

Tabelle 5.3: Kennziffern der Instanzen

Instanz	SH7_1	AMX_1	AMX_2	AMX_3
# Events	260	10460	7040	3614
# Dep	130	5228	3521	1809
# Arr	130	5232	3519	1805
# Activities	789	11983	8047	4094
# Drive	130	5205	3498	1786
# Wait	83	4595	3093	1585
# Change	576	2183	1456	723
# OD-Paare	44	2426	2426	2426
LB_2	927600.0	4.7509764E7	4.7509764E7	4.7509764E7
UB	927600.0	4.7690436E7	4.7690436E7	4.7692152E7
# virt. Knoten	16	102	102	102
# Origins	8	51	51	51
# Destinations	8	51	51	51
kleinstes L_a	60	18	18	18
größtes L_a	360	384	384	384
kleinstes U_a	120	36	36	36
größtes U_a	3720	3654	3654	3654
kleinstes Intervall	[120,120]	[18,36]	[18,36]	[18,36]
kleinste Länge Intervall	0	18	18	18
größtes Intervall	[180,3720]	[60,3654]	[60,3654]	[60,3654]
größte Länge Intervall	3540	3594	3594	3594
# Zsghskomp.	1	19	19	19
# Isolierte Knoten	0	2	2	2

Tabelle 5.4: Kennziffern der Instanzen

die Fahrpläne und Zielfunktionswerte nach Iteration 2 und in den folgenden Iterationen unterscheiden sich dementsprechend. Beim Testen mit Gurobi 3.0.0 wurden maximal 99 Iterationen durchgeführt.

Da `BellmanFord` nach den ersten Tests zu langsam schien, haben wir uns auf das Testen mit den Kürzeste-Wege-Methoden `TreeMapQueue` und `FibonacciHeap` beschränkt.

5.4 Analyse der Ergebnisse

Beim Vergleich der Ergebnisse von Gurobi 3.0.0 mit denen von Gurobi 4.5.1 wurden im Wesentlichen keine Unterschiede vom Verhalten der Heuristik festgestellt, so dass wir uns bei der Analyse der Ergebnisse auf die aus Version 4.5.1 beschränken werden (s. Tabelle 5.5).

Aus Zeitgründen haben wir uns bei den Tests mit Gurobi 4.5.1 auf maximal 50 Iterationen beziehungsweise einem Zeitlimit von 4 Stunden als Abbruchkriterium beschränkt.

Notation 5.4.1:
In den Spalten von Tabelle 5.5 stehen:

- ① die Instanz
- ② die Kürzeste-Wege-Methode und die Startlängen K_a
- ③ der Zielfunktionswert z, den die Heuristik beim Abbruch liefert
- ④ die Anzahl der Iterationen, die die Heuristik gebraucht hat
- ⑤ die Zeit, die die Heuristik gebraucht hat
- ⑥ der Zielfunktionswert nach 10 Minuten
- ⑦ die Zeit pro OD-Paar
- ⑧ die Abweichung vom Zielfunktionswert bei Abbruch zu LB_2 in %
- ⑨ der Zielfunktionswert in Iteration 1
- ⑩ die Abweichung vom Zielfunktionswert in Iteration 1 bei Abbruch zu LB_2 in %
- ⑪ der Zielfunktionswert in Iteration 2
- ⑫ die Abweichung vom Zielfunktionswert in Iteration 2 bei Abbruch zu LB_2 in %
- ⑬ Existiert ein Zykel (j/n), wenn ja in welchen Iterationen

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬
BG2H7_1	LA FH	4429478280	50	109 m 13 s	4430458920	0.0184	0.5021	4507707600	2.2771	4432660800	0.5743	n
	UA FH	4429929480	50	106 m 55 s	4431033000	0.0182	0.5123	4493189100	1.9477	4433026380	0.5826	n
	halb FH	4429850160	50	105 m 57 s	4431489000	0.0182	0.5105	4495696620	2.0046	4432588920	0.5727	n
	LA TMQ	4433332200	8	4 m 18 s	(**)	0.0016	0.5896	4480421460	1.6580	4434845100	0.6239	n
	UA TMQ	4437205560	6	3 m 58 s	(**)	0.0016	0.6774	4482665700	1.7089	4438596600	0.7090	n
	halb TMQ	4436287860	9	4 m 39 s	(**)	0.0016	0.6566	4482584340	1.7071	4439017680	0.7186	n
BGX_1	LA FH	20986486200	16	244 m 51 s	(*)	0.0382	2.9197	22288728480	9.3061	21025558620	3.1113	n
	UA FH	20983552860	16	243 m 5 s	(*)	0.0379	2.9053	22149813540	8.6248	21024714000	3.1072	n
	halb FH	20983911000	16	247 m 27 s	(*)	0.0385	2.9071	22101365400	8.3872	21025412220	3.1106	n
	LA TMQ	20997502320	8	23 m 14 s	21047553120	0.0048	2.9738	22460517540	10.1485	21047553120	3.2192	n
	UA TMQ	20991771420	50	125 m 23 s	21040417620	0.0055	2.9457	22275628680	9.2418	21040417620	3.1842	j, 9 & 11
	halb TMQ	20995899660	50	127 m 47 s	21041526960	0.0055	2.9659	22242474540	9.0792	21041526960	3.1897	j, 11 & 13
BGX_2	LA FH	4683030720	50	53 m 9 s	4683352560	0.0175	0.5888	4749025140	2.0063	4688417400	0.7045	n
	UA FH	4683230940	50	53 m 47 s	4683782820	0.0177	0.5931	4738762140	1.7859	4688280420	0.7015	n
	halb FH	4682827920	50	54 m 35 s	4682907180	0.0180	0.5844	4729697520	1.5912	4687307280	0.6806	n
	LA TMQ	4683507240	8	3 m 50 s	(**)	0.0019	0.5990	4756930200	2.1761	4684663080	0.6238	n
	UA TMQ	4684025460	8	3 m 33 s	(**)	0.0020	0.6101	4748672760	1.9987	4686099780	0.6547	n
	halb TMQ	4684423740	6	3 m 12 s	(**)	0.0018	0.6187	4745947680	1.9402	4685174760	0.6348	n
BGX_3	LA FH	155280	2	0 m 22 s	(**)	0.0058	0.0000	155280	0.0000	155280	0.0000	n
	UA FH	155280	2	0 m 24 s	(**)	0.0051	0.0000	155280	0.0000	155280	0.0000	n
	halb FH	155280	2	0 m 24 s	(**)	0.0043	0.0000	155280	0.0000	155280	0.0000	n
	LA TMQ	155280	2	0 m 24 s	(**)	0.0023	0.0000	155280	0.0000	155280	0.0000	n
	UA TMQ	155280	2	0 m 25 s	(**)	0.0021	0.0000	155280	0.0000	155280	0.0000	n
	halb TMQ	155280	2	0 m 24 s	(**)	0.0020	0.0000	155280	0.0000	155280	0.0000	n
BG3X_1	LA FH	359220	2	0 m 21 s	(**)	0.0020	0.0000	359220	0.0000	359220	0.0000	n
	UA FH	359220	2	0 m 21 s	(**)	0.0021	0.0000	359220	0.0000	359220	0.0000	n
	halb FH	359220	2	0 m 21 s	(**)	0.0023	0.0000	359220	0.0000	359220	0.0000	n
	LA TMQ	359220	2	0 m 20 s	(**)	0.0005	0.0000	359220	0.0000	359220	0.0000	n
	UA TMQ	359220	2	0 m 21 s	(**)	0.0006	0.0000	359220	0.0000	359220	0.0000	n
	halb TMQ	359220	2	0 m 25 s	(**)	0.0059	0.0000	359220	0.0000	359220	0.0000	n
BK2X_1	LA FH	17220110040	26	241 m 11 s	(*)	0.0327	2.2495	18274075140	8.5077	17254801920	2.4555	n
	UA FH	17217155460	27	248 m 18 s	(*)	0.0324	2.2319	18155796900	7.8054	17254736040	2.4551	n
	halb FH	17222014860	27	248 m 24 s	(*)	0.0325	2.2608	18162730260	7.8466	17256310980	2.4644	n
	LA TMQ	17230524660	13	21 m 9 s	17237046600	0.0043	2.3113	18425397780	9.4062	17285883900	2.6400	n
	UA TMQ	17238717600	14	23 m 20 s	17240167260	0.0045	2.3600	18300719220	8.6659	17278339380	2.5952	n
	halb TMQ	17233388580	12	19 m 57 s	17237250180	0.0044	2.3283	18293895600	8.6254	17276007180	2.5814	n
BK2X_2	LA FH	4467298680	50	43 m 15 s	4467668400	0.0168	0.6978	4548520500	2.5286	4470704160	0.7745	n
	UA FH	4469347320	50	39 m 45 s	4469439420	0.0155	0.7439	4542688020	2.3971	4473025800	0.8268	n
	halb FH	4468171440	50	42 m 22 s	4469502600	0.0166	0.7174	4547877120	2.5141	4476149160	0.8973	n
	LA TMQ	4470131880	8	2 m 9 s	(**)	0.0015	0.7616	4554948540	2.6735	4472658600	0.8186	n
	UA TMQ	4470591300	9	2 m 19 s	(**)	0.0015	0.7720	4550971620	2.5838	4473759780	0.8434	n
	halb TMQ	4470607800	8	2 m 13 s	(**)	0.0016	0.7723	4550652060	2.5766	4473723720	0.8426	n
BK2H7_1	LA FH	17121171420	24	248 m 3 s	(*)	0.0363	1.9618	18050129880	7.4940	17158672500	2.1851	n
	UA FH	17119553100	24	249 m 34 s	(*)	0.0365	1.9521	18069327600	7.6083	17163448320	2.2135	n
	halb FH	17124323160	23	240 m 0 s	(*)	0.0367	1.9805	17964762540	6.9856	17164614720	2.2205	n
	LA TMQ	17135523540	13	23 m 37 s	17143306980	0.0046	2.0472	18146921280	8.0704	17178376920	2.3024	n
	UA TMQ	17131173720	23	42 m 1 s	17139288000	0.0052	2.0213	18083829240	7.6947	17172202440	2.2657	n
	halb TMQ	17129921100	10	18 m 33 s	17140735320	0.0045	2.0139	18045077400	7.4639	17175171000	2.2833	n
BK2H7_2	LA FH	4424656560	50	47 m 48 s	4425476820	0.0183	0.4707	4487103240	1.8887	4427958900	0.5457	n
	UA FH	4424718900	50	46 m 1 s	4424982540	0.0176	0.4722	4484896380	1.8386	4428297660	0.5534	n
	halb FH	4424811420	50	48 m 41 s	4425246480	0.0189	0.4743	4480766640	1.7448	4427000760	0.5240	n
	LA TMQ	4427163960	5	2 m 26 s	(**)	0.0017	0.5277	4483341900	1.8033	4428003360	0.5467	n
	UA TMQ	4427489160	7	2 m 41 s	(**)	0.0029	0.5351	4489089300	1.9338	4428886860	0.5668	n
	halb TMQ	4427731560	7	2 m 33 s	(**)	0.0025	0.5406	4489075620	1.9335	4428259620	0.5526	n
SH7_1	LA FH	927600	50	0 m 7 s	(**)	0.0011	0.0000	1156800	24.7089	946800	2.0699	n
	UA FH	927600	50	0 m 7 s	(**)	0.0011	0.0000	1068000	15.1358	927600	0.0000	n
	halb FH	927600	50	0 m 7 s	(**)	0.0012	0.0000	1074000	15.7827	927600	0.0000	n
	LA TMQ	927600	3	0 m 1 s	(**)	0.0022	0.0000	1023600	10.3493	927600	0.0000	n
	UA TMQ	927600	3	0 m 1 s	(**)	0.0020	0.0000	966000	4.1397	927600	0.0000	n
	halb TMQ	927600	3	0 m 1 s	(**)	0.0030	0.0000	966000	4.1397	927600	0.0000	n
AMX_1	LA FH	47690436	50	45 m 17 s	47690436	0.0209	0.3803	52189206	9.8494	47698908	0.3981	n
	UA FH	47690436	50	42 m 11 s	47690436	0.0194	0.3803	52301076	10.0849	47705610	0.4122	n
	halb FH	47690436	50	42 m 58 s	47690436	0.0194	0.3803	52349568	10.1870	47701998	0.4046	n
	LA TMQ	47867604	9	3 m 23 s	(**)	0.0065	0.7532	49469952	4.1259	47970276	0.9693	n
	UA TMQ	47866812	9	3 m 35 s	(**)	0.0070	0.7515	49432992	4.0481	47969862	0.9684	n
	halb TMQ	47866812	9	3 m 13 s	(**)	0.0065	0.7515	49432992	4.0481	47969862	0.9684	n
AMX_2	LA FH	47690436	50	27 m 1 s	47690436	0.0125	0.3803	52281978	10.0447	47697120	0.3944	n
	UA FH	47691324	50	27 m 5 s	47691324	0.0125	0.3822	51997848	9.4467	47699856	0.4001	n
	halb FH	47692140	50	27 m 3 s	47692140	0.0125	0.3839	51722430	8.8669	47697468	0.3951	n
	LA TMQ	47765592	7	1 m 36 s	(**)	0.0041	0.5385	49473264	4.1328	48115752	1.2755	n
	UA TMQ	47770488	6	1 m 24 s	(**)	0.0041	0.5488	49436304	4.0550	48181800	1.4145	n
	halb TMQ	47770488	6	1 m 23 s	(**)	0.0040	0.5488	49436304	4.0550	48181800	1.4145	n
AMX_3	LA FH	47714976	50	12 m 18 s	47714976	0.0057	0.4319	52218300	9.9107	47748516	0.5025	n
	UA FH	47695440	50	15 m 2 s	47695440	0.0066	0.3908	52055208	9.5674	47738094	0.4806	n
	halb FH	47693082	50	14 m 56 s	47693082	0.0066	0.3859	52547646	10.6039	47737008	0.4783	n
	LA TMQ	47770692	7	0 m 54 s	(**)	0.0023	0.5492	49473264	4.1328	49473264	4.1328	n
	UA TMQ	47765076	9	1 m 12 s	(**)	0.0022	0.5374	49436304	4.0550	48511440	2.1084	n
	halb TMQ	47765076	9	1 m 6 s	(**)	0.0023	0.5374	49436304	4.0550	48511440	2.1084	n

Tabelle 5.5: Ergebnisse der Heuristik unter Verwendung von Gurobi 4.5.1
(*) z konnte noch nicht gefunden werden
(**) Heuristik hat weniger als 10 min gebraucht

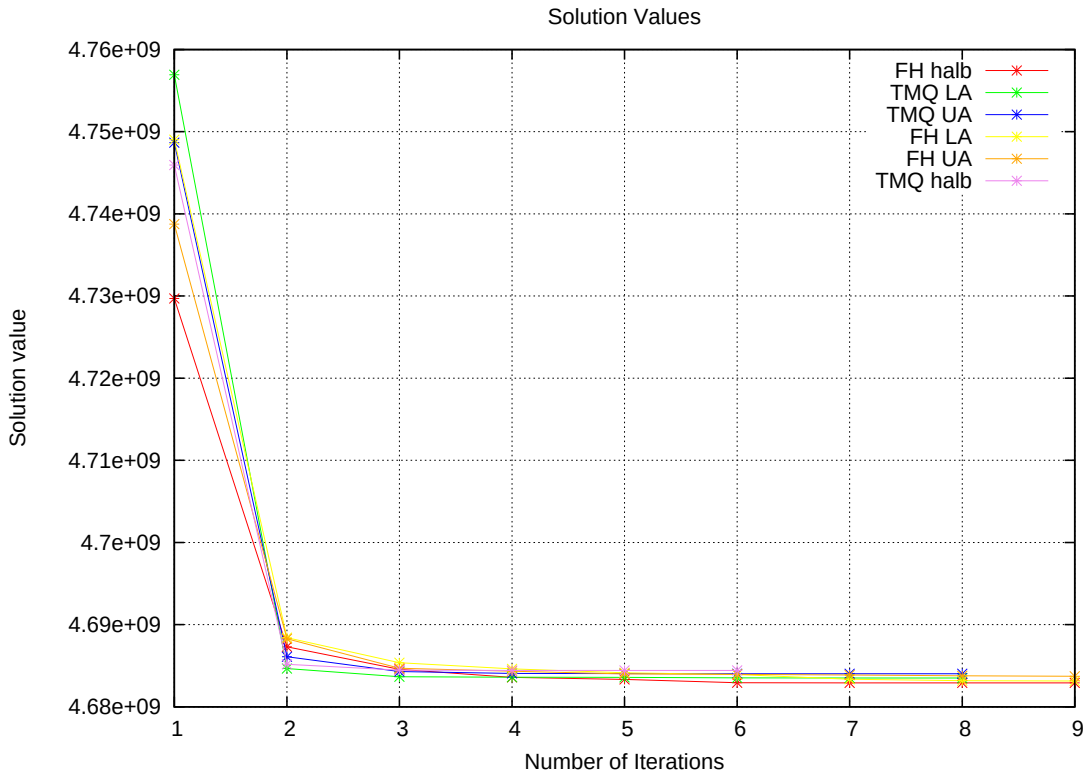


Abbildung 5.1: Plot der ersten 9 Zielfunktionswerte von BGX_2

Es stellte sich heraus, dass die Heuristik mit Kürzeste-Wege-Algorithmus `FibonacciHeap` bei Abbruch in vielen Fällen (vgl. dazu Tabelle 5.9) den besten Zielfunktionswert liefert. Dies gilt für alle drei Startlängen.

Eine erste Annahme, dass `LA` nach einer festgelegten Anzahl an maximalen Iterationen immer eine bessere Lösung liefert als `halb` und `UA`, stellte sich als falsch heraus. Ebensowenig konnte festgestellt werden, dass die Lösung von `halb` immer zwischen `LA` und `UA` lag, wie die Abbildungen 5.1 und 5.2 zeigen. Geplottet sind hier die ersten 9 Zielfunktionswerte. Betrachten wir allein die Plots von `TMQ`, so erkennen wir, dass in der ersten Iteration der Zielfunktionswert von `LA` größer als der von `UA` ist und dieser wiederum größer als der von `halb` ist. In der zweiten Iteration ist der Zielfunktionswert von `UA` jedoch größer als der von `halb` und der von `halb` größer als `LA`. `halb` bricht nach Iteration 6 ab mit einem Zielfunktionswert, der größer ist als der von `UA` und `LA` bei deren Abbruch in Iteration 8. Da bei allen Instanzen außer BGX_3, BG3X_1 und SH7_1 mit $(ILP5)$ beziehungsweise $(ILP8)$ in annehmbarer Zeit keine Optimallösung gefunden werden konnte, bietet es sich an die Abweichung des heuristischen Zielfunktionswertes zu einer unteren Schranke zu betrachten, um die Qualität der gefundenen Lösung zu überprüfen. Wie bereits in Kapitel 3.5 erwähnt, ist die größte unserer unteren Schranken LB_3. Um diese zu berechnen, müssen wir jedoch (Anzahl der OD-Paare)-mal das aperiodische Fahrplanproblem mit einem OD-Paar lösen. Wir haben dies für die Instanz BG2H7_1 getestet und die Berechnung war auch

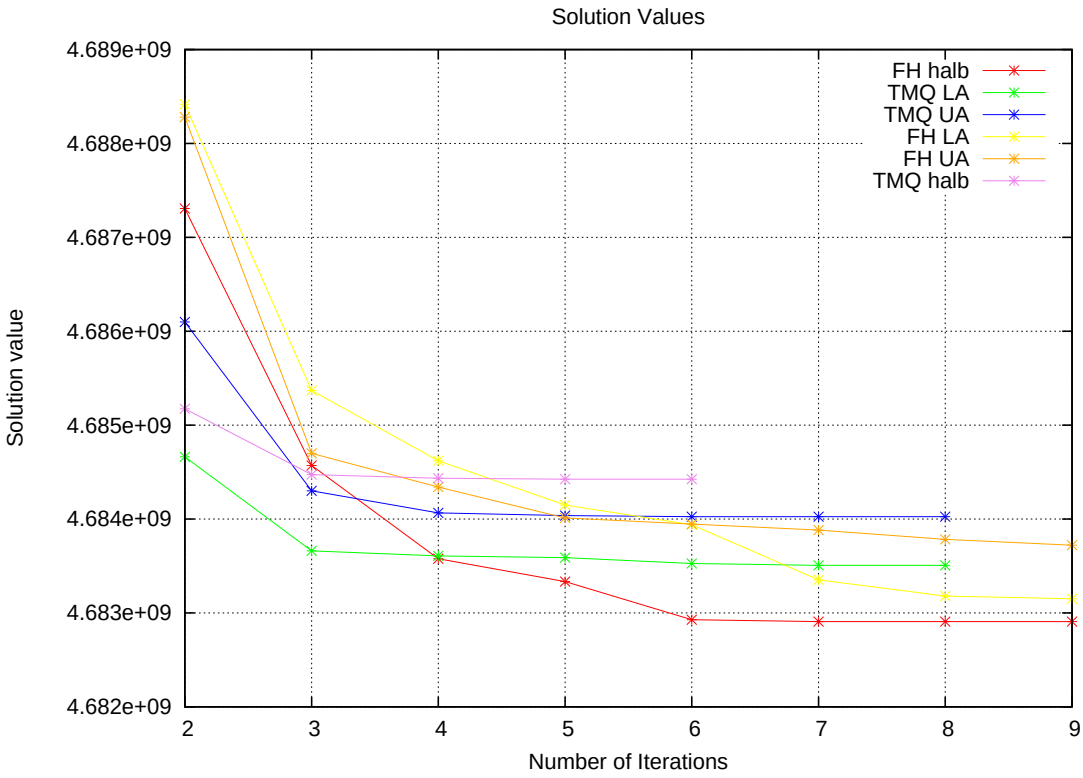


Abbildung 5.2: Plot der Zielfunktionswerte $2 - 9$ von BGX_2

nach drei Tagen nicht abgeschlossen. Aus diesem Grund haben wir für die Instanzen die untere Schranke LB_2 berechnet, was jeweils nur wenige Minuten gedauert hat, und vergleichen die Zielfunktionswerte, die die Heuristik liefert, jeweils mit der unteren Schranke LB_2 (s. Tabellen 5.2, 5.3 und 5.4).

Betrachten wir die Abweichung der Zielfunktionswerte bei Abbruch der Heuristik zur unteren Schranke LB_2, so fällt auf, dass die größte Abweichung 2.0738% beträgt. Bei den drei "größten" Instanzen BGX_1, BK2X_1 und BK2H7_1 ist auch die Abweichung am größten. Bei den drei Instanzen BGX_3, BG3X_1 und SH7_1 beträgt die Abweichung 0%, also wird hier sogar eine Optimallösung des aperiodischen Fahrplanproblems mit OD-Paaren gefunden. Die geringe Abweichung der Werte aller Instanzen zu LB_2 lässt darauf schließen, dass recht schnell eine gute Lösung gefunden werden kann.

Bestätigt wird diese Vermutung durch die Betrachtung der Abweichungen nach der ersten und zweiten Iteration. Während nach der ersten Iteration die Abweichung teilweise noch bis zu 24.7089% beträgt, ist die Abweichung nach Iteration 2 nur noch maximal 3.2192%. Die größte prozentuale Änderung in einer Iteration passiert also von der ersten auf die zweite Iteration. Alle darauffolgenden Änderungen sind im Vergleich dazu nur noch marginal (s. Abbildung 5.3). Es lohnt sich also - anders als wie zum Beispiel in LinTim bisher praktiziert - nicht nur eine Iteration auszuführen, sondern mindestens zwei.

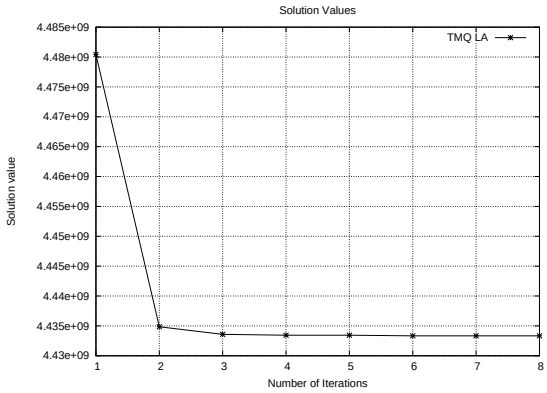
	∅ Abw. von z zu LB_2 nach 1. It in %	∅ Abw. von z zu LB_2 z nach 2. It in %	∅ Abw. von z zu LB_2 bei Abbruch in %	∅ Zeit pro OD-Paar in s
LA FH	6.8094	1.0551	0.8135	0.01759
UA FH	5.8648	0.9026	0.8123	0.01726
halb FH	5.8857	0.9037	0.8139	0.01745
LA TMQ	4.5136	1.3194	0.8962	0.00294
UA TMQ	3.8558	1.1777	0.9046	0.00316
halb TMQ	3.8172	1.1765	0.9026	0.00346
gesamt	5.1244	1.0892	0.8572	0.01031
∅ FH	6.1866	0.9538	0.8132	0.01743
∅ TMQ	4.0622	1.2245	0.9011	0.00319

Tabelle 5.6: Durchschnittswerte

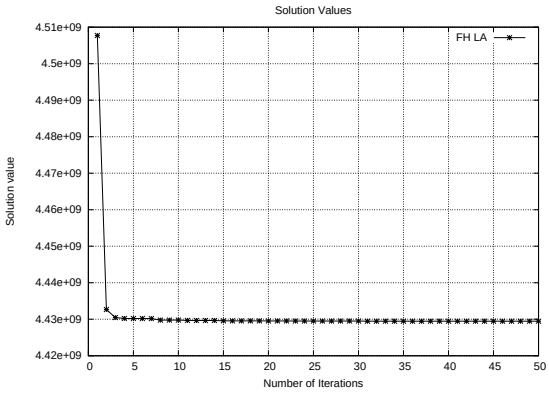
Auch die Betrachtung der Durchschnittswerte der Abweichungen der Zielfunktionswerte zur unteren Schranke LB_2 in Tabelle 5.6 stützt diese Behauptung. Während nach der ersten Iteration noch eine durchschnittliche Abweichung von 5.1244% besteht, beträgt diese nach Iteration 2 nur noch 1.0892%. Bei Abbruch der Heuristik ist sie nur noch 0.8572%. Die Aufteilung der Durchschnittswerte nach `FibonacciHeap` und `TreeMapQueue` zeigt, dass nach der ersten Iteration `FibonacciHeap` durchschnittlich einen schlechteren Zielfunktionswert hat als `TreeMapQueue`, aber nach der zweiten Iteration kehrt sich dieses Verhältnis um und bei Abbruch der Heuristik hat `FibonacciHeap` durchschnittlich einen besseren Zielfunktionswert als `TreeMapQueue`, wobei der Unterschied nicht mehr so groß ist wie nach der zweiten Iteration.

Charakteristisch für die getesten Instanzen ist auch, dass die Zeit pro OD-Paar und damit auch die Gesamtzeit, die die Heuristik benötigt, bei der `TreeMapQueue`-Methode kleiner ist (im Durchschnitt 0.00319 s pro OD-Paar), als bei `FibonacciHeap` (im Durchschnitt 0.01743 s pro OD-Paar). `FibonacciHeap` ist damit auch die Methode, bei der das zusätzlich eingeführte zeitliche Abbruchkriterium nach 4 Stunden gegriffen hat. Dazu trug auch die Tatsache bei, dass `FibonacciHeap` mehr Iterationen ausgeführt hat als `TreeMapQueue`. Während `TreeMapQueue` oft schon nach ≤ 10 Iterationen fertig war, hat `FibonacciHeap` oft das Limit von 50 Iterationen ausgenutzt.

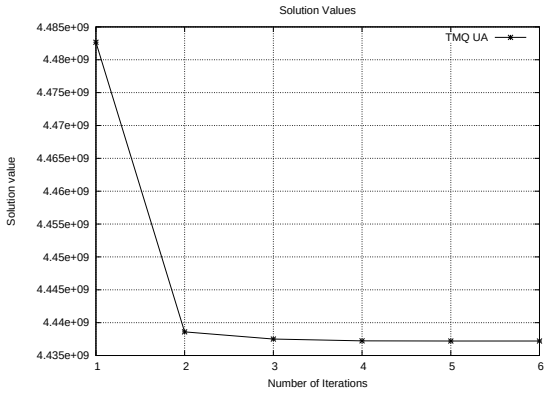
Beachtlich ist, dass bei der Instanz SH7_1 mit Kürzeste-Wege-Algorithmus `FibonacciHeap` in den ersten 50 Iterationen kein Zykel zu finden ist, obwohl beispielsweise mit Startlängen L_a ab Iteration 5 der Zielfunktionswert berechnet wird, der auch bei Abbruch ausgegeben wird und der der optimale Zielfunktionswert der aperiodischen Fahrplanproblems mit OD-Paaren ist. Also gibt es hier mindestens 45 unterschiedliche optimale Fahrpläne. Hieraus ergibt sich ein weiteres mögliches Abbruchkriterium, das wir aber nicht mit implementiert haben. Zusätzlich zum Überprüfen der Fahrpläne und der Zeit, könnte man also überprüfen, ob der Zielfunktionswert einer unteren Schranke (zum Beispiel LB_2) entspricht. Erklären könnte man



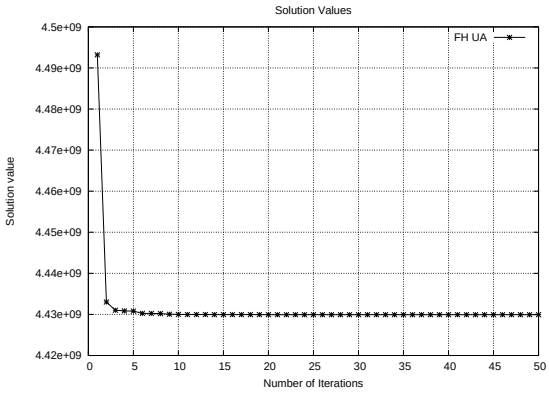
(a) LA TMQ



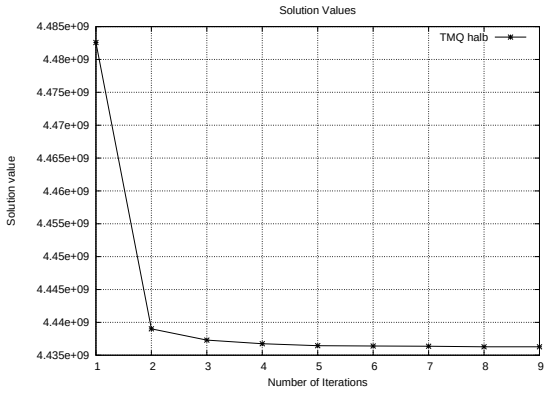
(b) LA FH



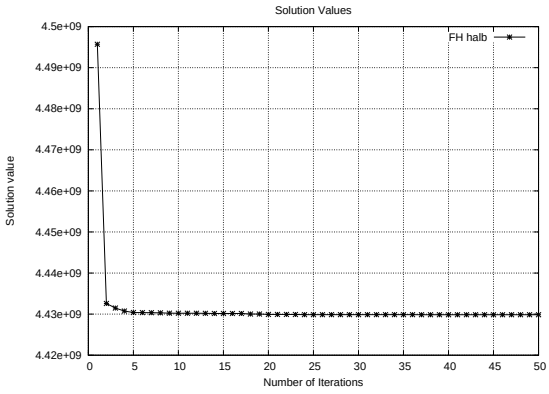
(c) UA TMQ



(d) UA FH



(e) halb TMQ



(f) halb FH

Abbildung 5.3: Plot von BG2H7_1 in allen 6 Konfigurationen

das Phänomen der vielen optimalen Fahrpläne mit der Struktur des Netzwerkes. Da die Instanzen mehr oder weniger künstlich erzeugt wurden und es durch das Ausrollen viele "parallele" Wege (das heißt Wege, die von demselben Abfahrtsbahnhof zum selben Ankunftsbahnhof mit den gleichen minimalen und maximalen Aktivitätsdauern gehen) gibt, gibt es auch mehr Optionen gleich gute Fahrpläne zu erzeugen. Deutlich wird dies auch in den schon erwähnten unterschiedlichen Ergebnissen von Gurobi 3.0.0 und Gurobi 4.5.1.

5.4.1 Warum funktioniert die Heuristik so gut?

Es stellt sich die Frage, warum die Heuristik bei den Tests so gut funktioniert, obwohl, wie in Lemma 3.4.9 gesehen, der relative Fehler beliebig groß werden kann. Die Antwort darauf kann nicht eindeutig gegeben werden. Vergleicht man die Tests jedoch mit dem Beispiel 3.5, so kann man mehrere Faktoren ausmachen, die einen Einfluss auf den relativen Fehler der Heuristik haben. Wir untersuchen nun also anhand von Beispiel 3.5, welche Faktoren den relativen Fehler beeinflussen.

- Die Intervalle $[L_a, U_a]$ $\forall a \in \mathcal{A}$ sind nicht beliebig groß.
  Je größer man in Beispiel 3.5 das w wählt, desto größer ist auch der relative Fehler.

- Das Verhältnis der Passagierzahlen der OD-Paare zueinander:
  Betrachten wir erneut das Beispiel aus Abbildung 3.5, aber diesmal mit den OD-Paaren $(u_2, v_3, 2)$, $(u_3, v_4, 1)$ und $(u_2, v_5, 1)$, so berechnet die Heuristik für jede unserer sechs Konfigurationen den richtigen optimalen Zielfunktionswert $z^* = 17$.

- Die Startlängen sind gut gewählt.
  Wir haben bereits im Beispiel aus Abbildung 3.5 gesehen, dass für die Startlängen `UA` und `halb` der optimale Zielfunktionswert richtig berechnet wurde, für `LA` jedoch nicht.

- In Tabelle 5.7 sind einige Fälle zu sehen, wo der relative Fehler bei kleinen Änderungen des Netzwerks aus Beispiel 3.5 nicht mehr beliebig groß ist. Das Ändern der Gewichte wurde bereits erwähnt. Vergrößern wir die untere Schranke L_a auf der Kante $(5, 6)$ auf 4 bei gleicher Gewichtung wie in Beispiel 3.5, so beträgt der relative Fehler der Heuristik 0% und wir erhalten somit für diesen Fall für alle getesteten Startlängen eine optimale Lösung des aperiodischen Fahrplanproblems mit OD-Paaren.

Wir sehen also, dass in Beispiel 3.5 viele Faktoren zusammenspielen müssen um einen großen relativen Fehler zu erhalten. Und auch kleinere Änderungen können bewirken, dass der Fehler nicht mehr beliebig groß wird.

Wir haben beobachtet, dass in Beispiel 3.5 bei Algorithmus 3.4.1 mit Startlängen `LA` und `halb` ein großer relativer Fehler auftritt. Für `UA` ist dies nicht der Fall. Wir können

jedoch ein Beispiel finden, bei dem dies auch für UA passiert. Wir betrachten dazu Abbildung 5.4. Wählen wir UA als Startlängen und berechnen die kürzesten Wege in der ersten Iteration, so bekommen wir für das OD-Paar den Weg $(3,4)$. Das heißt, das anschließende Timetabling wird die Länge dieses Weges so klein wie möglich machen, also auf die Länge w setzen. Die Länge des anderen Weges ist egal. Wird diese jetzt auf einen Wert $> w$ gesetzt, so findet der Kürzeste-Wege-Algorithmus in der nächsten Iteration wiederum den Weg $(3,4)$ und die Länge von $(1,2)$ ist abermals egal. Wenn der Solver die Länge dieses Weges nun auf den gleichen Wert $> w$ setzt wie in der Iteration davor, so bricht die Heuristik hier ab und wir bekommen als Zielfunktionswert $z^H = 1 \cdot w$. Der optimale Zielfunktionswert ist jedoch $z^* = 1$. Somit erhalten wir einen relativen Fehler von $err_{rel} := \frac{w-1}{1} \xrightarrow{w \to \infty} \infty$.

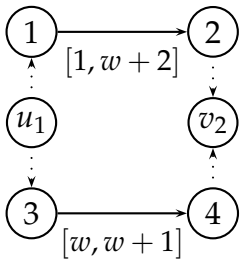


Abbildung 5.4: Ereignis-Aktivitätsnetzwerk mit OD-Paar $(u_1, v_2, 1)$

Und auch für die Startlängen $L_a \forall a \in \mathcal{A}$ kann es passieren, dass mit nur einem OD-Paar ein großer relativer Fehler auftritt. Wir betrachten dazu Abbildung 5.5 mit dem OD-Paar $(u_1, v_2, 1)$.

In der ersten Iteration werden mit Startlängen $L_a \ \forall a \in \mathcal{A}$ für OD-Paar $(u_1, v_2, 1)$ die kürzesten Wege $(2,3)$ und $(7,8)$ gefunden. Wird nun vom Kürzeste-Wege-Algorithmus der Weg $(2,3)$ ausgewählt, so wird im anschließenden Timetabling-Schritt versucht die Länge dieser Kante möglichst klein zu halten. Da aber nur die Länge w eine zulässige Lösung bietet, wird diese genommen. Die Länge der Kante $(7,8)$ ist egal. Wenn der Solver auch hier die Länge auf w setzt, so erhalten wir in der zweiten Iteration die kürzesten Wege $(2,3)$,$(5,6)$ und $(7,8)$. Wird eine der ersten beiden Kanten ausgewählt und im nächsten Timetabling die Kante $(7,8)$ wiederum auf w gesetzt, so bricht die Heuristik ab und gibt w als Zielfunktionswert zurück. Der optimale Zielfunktionswert beträgt jedoch 1, wodurch wir wieder einen relativen Fehler von $err_{rel} = \frac{w-1}{1} = w \xrightarrow{w \to \infty} \infty$ erhalten. Würden wir als Startlängen $U_a \ \forall a \in \mathcal{A}$ setzen, so würden wir die gleiche Situation wie oben beschrieben in Iteration 2 erhalten können und auch hier würde der relative Fehler beliebig groß werden.

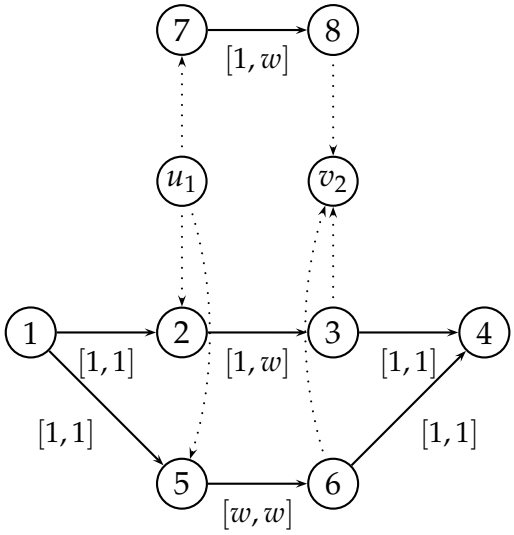


Abbildung 5.5: Ereignis-Aktivitätsnetzwerk mit OD-Paar $(u_1, v_2, 1)$

Kommen wir nun zurück zu den Tests mit den Instanzen aus Kapitel 5.2. Da für alle unsere Tests schon die untere Schranke LB_2 recht groß war, aber die Intervalle und auch die Passagierzahlen der OD-Paare im Vergleich dazu recht klein waren, haben wir hier nur kleine Abweichungen.

Warum der relative Fehler bei den Tests nicht beliebig groß wurde, kann auf Grund der vielen Faktoren, die sich gegenseitig auch wieder aufheben können (wie in Tabelle 5.7 zu sehen), nicht abschließend geklärt werden. Jedoch scheint schon die Nähe zur unteren Schranke LB_2 darauf hinzudeuten, dass die Kantenlängen vieler Wege der OD-Paare nahe an L_a für alle $a \in \mathcal{A}$ liegen (s. Tabelle 5.8). Das stichprobenartige Überprüfen bestätigte diese Vermutung. Für die Instanz AMX_3 mit schlechtestem heuristischem Zielfunktionswert 47770692 erhalten wir durch Lösen aller OD-Paare einzeln (also durch das Lösen von $(ILP5)$ beziehungsweise $(ILP8)$ mit einem OD-Paar aus der OD-Menge und dies für alle OD-Paare durchgeführt und alle Zielfunktionswerte aufsummiert) eine untere Schranke von $LB_3 = 47670762$ gerundet (der tatsächliche Wert beträgt durch die Ungenauigkeit von Gurobi $LB_3 = 47670761,9826672$) was einer Abweichung von 0.209% entspricht. So ist zum Beispiel der Zielfunktionswert $z^1(17,1)$, wenn man das aperiodische Fahrplanproblem mit Instanz AMX_3 und OD-Paar $(17,1)$ löst, $z^1(17,1) = 26442$. Löst man AMX_3 mit allen 2426 OD-Paaren und guckt sich den zum OD-Paar $(17,1)$ passenden Summanden an, so enspricht dieser $z^{OD}_{\Pi^H}(17,1) = 26676$ für den durch Algorithmus 3.4.1 berechneten Fahrplan Π^H. Bei 329 OD-Paaren wurde eine Abweichung dieser beiden Werte festgestellt. Also fahren 2097 OD-Paare so, wie sie fahren würden, wenn sie alleine einflussgebend wären, wobei auch noch 99 OD-Paare dabei waren, wo Gurobi binäre Variablen nicht binär gesetzt hat und also ein nicht-ganzzahliger Wert berechnet wurde (s. auch Kapitel 5.5).

$w_{u_2v_3}$	$w_{u_3v_4}$	$w_{u_2v_5}$	Wege Heuristik nicht opt.	z_H & K_a	opt. Wege	z^*	andere Änderungen
1	3	1	$(5,6)$	5015 LA	$(8,9)$	21	
1	2	1	$(5,6)$	5013 LA	$(8,9)$	18	
1	1	1	$(5,6)$	5011 LA	$(8,9)$	15	
2	1	1				17	
1	1	2	$(5,6)$	5021 LA	$(8,9)$	25	
1	1	1000	$(5,6)$	15001 LA	$(8,9)$	10005	
1	3	1	$(5,6)$	5017 LA	$(8,9)$	21	$(5,6,[3,w])$
1	3	1				21	$(5,6,[4,w])$

Tabelle 5.7: Untersuchungen bei kleinen Änderungen des Ereignis-Aktivitätsnetzwerkes aus Abbildung 3.5 mit $w = 5000$

Instanz	#OD	z^*	$[z^*]$	größtes z^H	$\#z^1(u,v) = z^H(u,v)$	$\#z^1(u,v) \notin \mathbb{Z}$	∅ Abw. Heuristik OD-Paar
AMX_3	2426	47670761,98	47670762	47770692	2097	99	41,19
BGX_2	3197	4660295467	4660295467	4684423740	2679	1	7547,16

Tabelle 5.8: Vergleich der Reisezeiten beim exakten Berechnen mit nur einem OD-Paar und der einzelnen Reisezeiten der OD-Paare bei heuristischer Berechnung mit allen OD-Paaren

Berechnet man für alle OD-Paare einzeln mit Hilfe der Heuristik mit Startlängen $L_a \ \forall a \in \mathcal{A}$ einen Zielfunktionswert und summiert die einzelnen Zielfunktionswerte auf, so bekommt man einen größeren Zielfunktionswert, als wenn das aperiodische Fahrplanproblem mit allen OD-Paaren und denselben Startlängen heuristisch gelöst wird. So erhalten wir für das OD-Paar $(17,1)$ etwa einen heuristischen Zielfunktionswert von $z^H = 28470$. Und auch wenn man für dieses OD-Paar die Wege vergleicht, so sieht man, dass alle drei Varianten unterschiedliche Wege benutzen. Da es alleine 77 Abfahrtsereignisse und 8 Ankunftsereignisse gibt, ist es nicht möglich in annehmbarer Zeit herauszufinden, wo die Ursache hierfür liegt.

Da wir alle Testinstanzen aus LinTim genommen haben und diese jeweils durch das Ausrollen viele parallele Wege haben, könnte es sein, dass auch diese für die guten Ergebnisse eine Rolle spielen.

5.5 Vergleich der ganzzahligen linearen Programme mit der Heuristik

Nun werden wir untersuchen, wie gut beziehungsweise schlecht $(ILP5)$ und $(ILP8)$ in der Praxis sind und ob es tatsächlich eine Rechtfertigung gibt, Algorithmus 3.4.1 zu verwenden. Dazu haben wir uns einige Testinstanzen genommen und heuristisch sowie mit $(ILP5)/(ILP8)$ und $(ILP4)/(ILP6)$ den Zielfunktionswert berechnet. Als untere Schranke für $(ILP5)$ und $(ILP8)$ wurde jeweils LB_2 genommen. Wir werden in diesem Abschnitt lernen, welches ganzzahlige lineare Programm sich mit welchem M am "besten" zum Berechnen der Optimallösung eignet. Die durch $(ILP2)/(ILP3)$ und $(ILP7)$ berechneten M's sind in Tabelle A.1 zu finden.

Es fiel auf, dass $(ILP4)$ und $(ILP6)$ (also die Programme ohne untere Schranke LB) schon bei den ersten vier Instanzen (mit OD-Datei `ChosenOD_1034`, `ChosenOD_2828`, `ChosenOD_4222` und `ChosenOD_7231`) nicht in der Lage waren, eine Optimallösung zu berechnen beziehungsweise zu beweisen, dass die gefundene Lösung eine Optimallösung ist. Für die anderen Instanzen konnten keine vernünftige Ergebnisse berechnet werden. Die Zeiten und berechneten Zielfunktionswerte (nicht unbedingt die optimalen) sind in Tabelle A.4 abzulesen. Da man die heuristische Lösung für diese Instanzen also nicht mit einer von $(ILP4)$ und $(ILP6)$ berechneten Optimallösung vergleichen kann, beschränken wir uns im Weiteren auf das Vergleichen der heuristischen Lösung mit der von $(ILP5)$ und $(ILP8)$.

Tabelle 5.9 zeigt die durch $(ILP5)$ und $(ILP8)$ berechneten Zielfunktionswerte von 7 verschiedenen Instanzen, wobei alle auf demselben Ereignis-Aktivitätsnetzwerk beruhen, aber unterschiedliche OD-Mengen haben. Außerdem ist die Zeit für das Berechnen der Zielfunktionswerte abzulesen. Es sind für jede Konfiguration fünf Zeiten angegeben. Die oberste ist die Zeit für das Berechnen der Ms. Die zweite die Zeit, die $(ILP5)$ beziehungsweise $(ILP8)$ braucht, bis es die ausgegebene Lösung gefunden hat. Die dritte Zeit gibt die Nettozeit (ohne Aufstellen des ganzzahligen linearen Programms) für das Lösen der ganzzahligen linearen Programme an, die vierte die Zeit inklusive Aufstellen des ganzzahligen linearen Programms und schließlich die fünfte die Gesamtzeit. Für das Berechnen der ganzzahligen linearen Programme wurde ein Zeitlimit von 2 Stunden gegeben und die Zielfunktionswerte sind entsprechend die Werte, die nach dem Überschreiten dieses Limits ausgegeben wurden. Das Aufstellen der ganzzahligen linearen Programme konnte mehr als zwei Stunden dauern. Zum Vergleich stehen in Tabelle 5.11 die Zielfunktionswerte, die Algorithmus 3.4.1 berechnet hat, sowie die Anzahl der Iterationen (wiederum maximal 50 Iterationen) und die benötigte Zeit.

Formal heißt das:

Notation 5.5.1:
In Tabelle 5.9 bezeichnen wir mit :

- z den Zielfunktionswert, den $(ILP5)$ beziehungsweise $(ILP8)$ bei Abbruch nach 2 Stunden liefert

- t die Zeiten in Sekunden, wobei in Reihe
  - 1 die Zeit zum Berechnen von M_e^1, M_e^2, M_v^1 oder M_v^2 steht (inklusive Aufstellen von $(ILP2)$ und $(ILP3)$ bei M_e^2 beziehungsweise $(ILP7)$ bei M_v^2)
  - 2 die Zeit steht, bei der der Zielfunktionswert z gefunden wurde
  - 3 die Zeit steht, die $(ILP5)$ beziehungsweise $(ILP8)$ ohne Aufstellen des jeweiligen ILPs gebraucht hat
  - 4 die Zeit steht, die $(ILP5)$ beziehungsweise $(ILP8)$ mit Aufstellen des jeweiligen ILPs gebraucht hat
  - 5 die Zeit steht, die das Lösen des jeweiligen ILPs insgesamt gebraucht hat

In Tabelle 5.10 sind die unteren Schranken LB_2 für jede der genannten Instanzen zu finden.

Da die Variablenanzahl schnell wächst, wenn die Anzahl der OD-Paare und die Anzahl der möglichen Wege für ein OD-Paar steigen, stellte sich heraus, dass es nur in sehr wenigen Fällen möglich ist die Optimallösung mit $(ILP5)$ und/oder $(ILP8)$ zu berechnen beziehungsweise zu beweisen, dass der aktuelle Zielfunktionswert optimal ist. Die Anzahl an Variablen sowie Nebenbedingungen sind in Tabelle A.5 bis Tabelle A.8 zu finden. Beim Aufstellen wurden für die Ereignisse, die die einzigen Knoten in einer Zusammenhangskomponente waren, die entsprechenden $q-$Variablen für die Origin- beziehungsweise Destination-Kante nicht mit berücksichtigt, da sie ohnehin auf 0 gesetzt werden würden. Aus diesem Grund gibt es zum Beispiel für die Instanz mit OD-ID 1034 (mit 2 isolierten Knoten) nicht #Π-Variablen $\cdot$ # OD-Paare $= 3614 \cdot 5 = 18070$ Flusserhaltungsgleichungen, sondern nur $3612 \cdot 5 = 18060$ von diesen.

Man erkennt in Tabelle 5.11 und Tabelle 5.9, dass die Heuristik bei "kleinen" Instanzen schon nach kurzer Zeit eine gute Genauigkeit liefert. Die Instanzen mit den OD-IDs 1034 und 2828 finden sogar eine Optimallösung, welche der jeweiligen unteren Schranke LB_2 entspricht. Bei den Instanzen mit den OD-IDs 8939 und 7859 könnte es sein, dass eine Optimallösung gefunden wurde (bei 8939 mit `LA FH` und `halb FH`, bei 7859 mit `UA FH`). Zumindestens war der gefundene heuristische Wert bei beiden Instanzen genauso gut wie der berechnete Wert von $(ILP8)$ mit M_v^1 und M_v^2. Allerdings hat es $(ILP8)$ sowohl mit M_v^1 als auch mit M_v^2 nicht geschafft, zu beweisen, dass es keine bessere Lösung gibt und der berechnete Zielfunktionswert somit auch wirklich eine Optimallösung ist.

Bei 4222 wurden trotz des expliziten Deklarierens der Variablen q_a als binäre Variablen teilweise Werte für diese berechnet, die zwar nahe an 1 oder 0 liegen, aber trotzdem keine binären Variablen sind (zum Beispiel 0.9999928855150891). Das kommt daher, dass Gurobi für seine Variablen Toleranzen gesetzt hat

Instanz	#OD	$(ILP5)$				$(ILP8)$			
		M_e^1		M_e^2		M_v^1		M_v^2	
		z	t	z	t	z	t	z	t
1034	5	56898_H	0 394 394 790 791	56898_H	14753 325 325 741 15496	56898_*	0 8 8 13 14	56898_*	314 1808 1808 1812 2128
2828	7	112578_H	0 2964 7200 8081 8083	94320_H	36406 101 101 939 37347	94320_H	0 18 18 25 26	94320_H	448 60 60 46 515
4222	8	169164_H	0 816 816 1901 1902	169428_H	9544 1371 7200 8307 17852	169428_H	0 99 7200 7205 7206	169428_H	318 591 7200 7203 7523
7231	10	191344_H	0 1198 7200 9120 9122	197892_H	28168 3263 7200 8952 37121	191340_H	0 437 437 461 462	191340_H	715 285 285 293 1010
8939	15	453696_H	0 4004 7200 11258 11259	499746_H	29966 2018 7200 11312 41280	451626_H	0 179 7200 7215 7217	451626_H	755 3262 7200 7208 7964
7859	15	201936_H	0 2259 7200 11033 11034	227622_H	6982 3736 7200 10890 17874	197058_H	0 416 7200 7213 7215	197058_H	810 1693 7200 7205 8016
7196	17	506940_H	0 6423 7200 12619 12621	452004_H	38225 5942 7200 11918 50145	455574_H	0 116 7200 7216 7218	457290_H	858 47 7200 7213 8073

Tabelle 5.9: Zielfunktionswerte von $(ILP5)$ und $(ILP8)$ uner Verwendung von LB_2
_H - Wert durch Heuristik von Gurobi berechnet
_* - Wert durch Branching von Gurobi berechnet
Das Basis-Ereignis-Aktivitätsnetzwerk war AMX_3 mit 3614 Knoten und 4094 Kanten. Mit der OD-Datei OD_neu_386.txt als Basis wurden unabhängig voneinander verschiedene Teilmengen der OD-Paare ausgewählt und getestet.

Instanz	1034	2828	4222	7231	8939	7859	7196
LB_2	56898	94320	169164	191340	449772	196938	451068
LB_3	56898	94320	169428	191340	451626	197058	452004

Tabelle 5.10: Die unteren Schranken LB_2 und LB_3 für die Instanzen aus 5.9

(s. http://groups.google.com/group/gurobi/browse_frm/thread/464bce71a21c341e#). Aus diesem Grund ist der berechnete Zielfunktionswert für Instanz 4222 $(ILP5)$ mit M_e^1 falsch.

Vergleicht man die Zeiten, die die Heuristik zum Lösen der Instanzen gebraucht hat, mit der Dauer der ganzzahligen linearen Programme, so erkennt man auch hier, dass es sich lohnt die Heuristik zu benutzen. Während die Heuristik schon nach wenigen Sekunden mit der Berechnung fertig war, brauchten die ganzzahligen linearen Programme teilweise mehr als zwei Stunden.

Da der optimale Zielfunktionswert beim Lösen der von $(ILP5)$ und $(ILP8)$ bei den Instanzen 1034 und 2828 mit der unteren Schranke LB_2, die in diesen Fällen auch LB_3 ist, übereinstimmt, haben wir LB_3 auch für die anderen Instanzen berechnet und wir können erkennen, dass diese auch bei den anderen Instanzen angenommen werden kann, die Optimalität jedoch nicht bewiesen wird. Daraufhin haben wir die ganzzahligen linearen Programme der Instanzen noch einmal unter Verwendung von LB_3 gelöst (da, wo LB_2 mit LB_3 nicht schon übereinstimmt). Die Ergebnisse sind in Tabelle 5.12 zu finden. Die ersten beiden Spalten geben dabei die OD-ID und die Anzahl der OD-Paare an und die Spalten $3-6$ und $7-10$ die Zielfunktionswerte z und Zeit t, in der das jeweilige ganzzahlige lineare Programm gelöst wurde. Dabei wurde bei allen Instanzen außer 7196 unsere Vermutung bestätigt, dass der exakte Zielfunktionswert mit der unteren Schranke LB_3 übereinstimmt, die Passagiere also exakt den Weg nehmen, den sie auch nehmen würden, wenn sie einzeln fahren würden. Wir können aus Tabelle 5.9 und 5.10 weiterhin ablesen, dass die untere Schranke LB_3 auch für Instanz 7196 der optimale Zielfunktionswert ist.

Damit stimmt bei allen 7 Instanzen der optimale Zielfunktionswert mit der unteren Schranke LB_3 überein.

Das Berechnen durch die Kanten-basierende Methode $(ILP5)$ dauert länger, als wenn man die virtuelle-Kanten-basierende Methode (IL8) nimmt. Schon für die Instanz 2828 mit 7 OD-Paaren war $(ILP5)$ mit M_e^1 nicht in der Lage eine Optimallösung zu berechnen, wohingegen $(ILP5)$ mit M_e^2 und $(ILP8)$ mit M_v^1 und M_v^2 eine Optimallösung mit Zielfunktionswert $z = 94320$ berechneten. Dabei war die virtuelle-Kanten-basierende Variante $(ILP8)$ schneller als $(ILP5)$. Dies bestätigt sich auch bei allen anderen Instanzen außer 1034, wo $(ILP5)$ schneller war als $(ILP8)$ mit M_v^2. Generell kann man also erkennen, dass es sich lohnt $(ILP8)$ zu verwenden. Der Aufwand um M_v^2 zu berechnen rentierte sich in diesen Fällen allerdings nicht.

Sowohl $(ILP5)$ als auch $(ILP8)$ brauchen mit M_e^2 beziehungsweise M_v^2 zwar weniger Zeit zum Berechnen des LPs, aber schon für das Berechnen der M selbst ist ein großer

ID OD	z_1^H	t_1^H	z_2^H	t_2^H	z^H
1034	56898	3	56898	5	56898 (LA TMQ)
	56898	2	56898	4	56898 (UA TMQ)
	56898	2	56898	5	56898 (halbTMQ)
	67482	3	56898	5	56898 (LA FH)
	60846	3	60174	5	60174 (UA FH)
	57570	2	57570	4	57570 (halb FH)
2828	94320	3	94320	5	94320 (LA TMQ)
	94320	3	94320	5	94320 (UA TMQ)
	94320	2	94320	5	94320 (halb TMQ)
	108180	2	94320	4	94320 (LA FH)
	94320	2	94320	4	94320 (UA FH)
	110280	3	94320	5	94320 (halb FH)
4222	172638	3	172638	4	172638 (LA TMQ)
	172638	2	172638	5	172638 (UA TMQ)
	172638	3	172638	5	172638 (halb TMQ)
	173772	3	170028	5	170028 (LA FH)
	173772	2	173772	4	173772 (UA FH)
	175644	2	173772	4	173772 (halb FH)
7231	192408	3	192408	5	192408 (LA TMQ)
	192408	9	192408	12	192408 (UA TMQ)
	192408	3	192408	5	192408 (halb TMQ)
	196368	3	191604	5	191604 (LA FH)
	199668	3	192210	5	192210 (UA FH)
	199350	3	191754	5	191754 (halb FH)
8939	461526	3	454860	5	454860 (LA TMQ)
	461526	2	454860	5	454860 (UA TMQ)
	461526	3	454860	5	454860 (halb TMQ)
	462468	3	451626	5	451626 (LA FH)
	470748	3	456876	5	456876 (UA FH)
	464652	3	451626	5	451626 (halb FH)
7859	208620	3	198048	6	198048 (LA TMQ)
	208620	3	198048	5	198048 (UA TMQ)
	208620	3	198048	5	198048 (halb TMQ)
	222978	3	200298	5	200298 (LA FH)
	229176	3	202908	5	197058 (UA FH)
	237240	3	206148	5	206148 (halb FH)
7196	460644	3	460224	5	453558 (LA TMQ)
	460644	3	460224	5	453558 (UA TMQ)
	460644	3	460224	5	453558 (halb TMQ)
	468564	3	453096	6	453096 (LA FH)
	471414	3	455442	5	453096 (UA FH)
	470370	3	454944	5	454944 (halb FH)

Tabelle 5.11: Die heur. Werte der 1. und 2. Iteration aus Tabelle 5.9
z_1^H - Zielfunktionswert in Iteration 1, t_1^H - Zeit in Iteration 1 in s
z_2^H - Zielfunktionswert in Iteration 2, t_2^H - Zeit in Iteration 2 in s
z^H - Zielfunktionswert bei Abbruch

Instanz	#OD	$(ILP5)$				$(ILP8)$			
		M_e^1		M_e^2		M_v^1		M_v^2	
		z	t	z	t	z	t	z	t
4222	8	169428	4710	179406	7200	169428	1304	169428	1304
8939	15	n.A.	n.A.	452970	7200	451626	875	451626	3185
7859	15	n.A.	n.A.	197058	3223	197058	1796	197058	237
7196	17	498678	7200	454380	7200	454242	7200	455298	7200

Tabelle 5.12: Zielfunktionswerte von AMX_3 mit unterschiedlichen OD-Mengen und $(ILP5)$ und $(ILP8)$ unter Verwendung von LB_3
n.A. - Instanz wurde vor dem Zeitlimit ohne Lösung abgebrochen

Zeitaufwand erforderlich. Um diesen zu verkürzen und immer noch geeignete M zu berechnen, ist es sinnvoll nicht nur für das Berechnen des LPs ein Zeitlimit zu setzen, sondern auch für das Berechnen der M und, falls das Zeitlimit überschritten wurde, für alle noch nicht berechneten M_e^2 bzw, M_v^2 die größeren M's M_e^1 beziehungsweise M_v^1 zu verwenden.

Notation 5.5.2:
In Tabelle 5.13 bezeichnen wir mit :

- z den Zielfunktionswert, den $(ILP5)$ beziehungsweise $(ILP8)$ bei Abbruch liefert
- t die Zeiten in Sekunden, wobei in Reihe
  - 1 die Zeit zum Berechnen von M_e^1, M_e^2, M_v^1 oder M_v^2 steht (inklusive Aufstellen von $(ILP2)$ und $(ILP3)$ bei M_e^2 beziehungsweise $(ILP7)$ bei M_v^2)
  - 2 die Zeit steht, die $(ILP5)$ beziehungsweise $(ILP8)$ ohne Aufstellen des jeweiligen ILPs gebraucht hat
  - 3 die Zeit steht, die $(ILP5)$ beziehungsweise $(ILP8)$ mit Aufstellen des jeweiligen ILPs gebraucht hat
  - 4 die Zeit steht, die das Lösen des jeweiligen ILPs insgesamt gebraucht hat

Es scheint von der Struktur des Netzwerkes abzuhängen, ob in annehmbarer Zeit eine Lösung gefunden werden kann oder nicht. Bei den Instanzen aus Tabelle 5.9 ist es teilweise schon schwer für 8 OD-Paare eine Lösung zu finden. Betrachten wir hingegen die Ergebnisse aus Tabelle 5.13, so sehen wir, dass es hier ohne Probleme für beide Instanzen möglich ist, eine Optimallösung zu berechnen. Dies ist trotz der ungefähr gleichen (BGX_3) oder höheren (BG3X_1) Anzahl an Variablen und Nebenbedingungen möglich. Betrachten wir die Tabellen mit der Anzahl der Nebenbedingungen und Variablen genauer, so erkennen wir, dass es für die sieben Instanzen aus Tabelle 5.9 eine deutlich höhere Anzahl an Wegen (von durchschnittlich ca. 75-130, s. Tabelle A.6)

gibt, als für die beiden Instanzen aus Tabelle 5.13, die jeweils nur durchschnittlich ca. 1 bis 2 mögliche Wege pro OD-Paare besitzen (s. Tabelle A.10). Die Auswahl an Wegen ist hier also deutlich geringer und damit auch einfacher.

Vergleicht man die Zielfunktionswerte, die die Heuristik für `BG3X_1` und `BGX_3` liefert (s. Tabelle 5.5) und die optimalen Zielfunktionswerte, die $(ILP5)$/$(ILP8)$ berechnet (s. Tabelle 5.13), so erkennt man, dass die Heuristik für diese Instanzen sogar den exakten Wert berechnet hat. Auch hier könnte die Ursache in der geringen Auswahlmöglichkeit der Wege liegen.

Instanz	#OD	$(ILP5)$				$(ILP8)$			
		M_e^1		M_e^2		M_v^1		M_v^2	
		z	t	z	t	z	t	z	t
BG3X_1	213	359220_H	0 2.14 77092 77095	359220_H	68 1.87 103164 103236	359220_H	0 0.07 2 6	359220_H	112 0.07 2 118
BGX_3	29	155280_*	0 0.49 2233 2234	155280_*	9 0.57 2077 2088	155280_*	0 0.05 1 2	155280_*	12 0.04 1 15

Tabelle 5.13: Die Ergebnisse von $(ILP5)$ und $(ILP8)$ bei den Instanzen BGX_3 und BG3X_1

6 Fazit

Da das aperiodische Fahrplanproblem mit OD-Paaren $\mathcal{NP}$-schwer ist, ist es nicht für alle Instanzen möglich, die Optimallösung innerhalb einer zumutbaren Zeit zu berechnen. Es wurde in dieser Arbeit eine iterative Heuristik vorgestellt, die einen zulässigen Fahrplan für das aperiodische Fahrplanproblem mit OD-Paaren gefunden hat, indem sie die beiden Teilprobleme Routing und Timetabling hintereinander ausgeführt und dieses iteriert hat. Gezeigt wurde, dass die berechneten Zielfunktionswerte in keinem Iterationsschritt schlechter waren als in der Iteration zuvor. Untersucht wurde die Heuristik in Hinblick auf Schnelligkeit und Genauigkeit mit verschiedenen Startlängen für den Anfang der Heuristik und verschiedenen Kürzeste-Wege-Methoden.

Es gab Fälle, in denen die Heuristik schon mit den Startlängen einen optimalen Fahrplan gefunden hat. Weiterhin konnte beobachtet werden, dass für einige Instanzen die Heuristik in ein lokales Optimum gelaufen ist.

Bei den meisten untersuchten Instanzen gab es eine Verbesserung des Zielfunktionswertes. Es konnte festgestellt werden, dass, falls es eine Verbesserung der Zielfunktionswerten gab, die größte Änderung nach der ersten Iteration stattfand. Aber auch danach gab es teilweise noch Änderungen, die es wert waren, diese Iterationen abzuwarten und nicht gleich nach dem ersten Iterationsschritt abzubrechen. Anders als bisher praktiziert, kann es sich also lohnen, mindestens zwei Iterationsschritte durchzuführen.

Weiterhin konnte festgestellt werden, dass es sinnvoll ist, unterschiedliche Startlängen und unterschiedliche Kürzeste-Wege-Methoden auszuprobieren. Es konnte kein eindeutiges Ergebnis erzielt werden, welche Startlängen und welche Kürzeste-Wege-Methode "am besten" sind. Betrachtet man in dieser Hinsicht alleine die Laufzeit, so war die Kürzeste-Wege-Methode `TREE_MAP_QUEUE` der Methode `FIBONACCI_HEAP` überlegen. In vielen Fällen lieferte `FIBONACCI_HEAP` dafür jedoch den besseren Zielfunktionswert. Dieses könnte jedoch von der Struktur des Netzwerkes abhängen und wäre damit ein interessantes Thema für weitere Forschungen auf diesem Gebiet.

Entgegen den Erwartungen und im Gegensatz zum Beispiel 3.5 mit dem großen relativen Fehler lieferte die Heuristik für die getesteten Instanzen sehr gute Zielfunktionswerte, die nahe an der unteren Schranke LB_2 und dementsprechend auch am optimalen Zielfunktionswert lagen. Man könnte nun untersuchen, ob sich dieses in der Praxis weiterhin so bestätigt, oder ob die durch LinTim erzeugten Instanzen eine Besonderheit aufweisen, die dieses verursachen.

Betrachtet man die Struktur der Netzwerke, bei denen es überhaupt Änderungen in

den Zielfunktionswerten gibt, so fällt auf, dass bei vielen Kanten die Dauer $[L_a, U_a]$ schon durch $U_a = L_a$ festgelegt ist. Untersuchen könnte man hierbei, ob man diese Kanten zu einem Cluster zusammenfassen kann um die Heuristik noch schneller zu machen.

In unseren Betrachtungen wurden die Headways ausgeschlossen. Interessant ist nun die Frage, ob die Headways einen Einfluss auf den Fahrplan haben. Zu beachten ist, dass die Headway-Aktivitäten im Routing-Prozess nicht mit betrachtet werden dürfen, da die Passagiere selbst keine Headways benutzen. Dementsprechend muss auch bei den ganzzahligen linearen Programmen aufgepasst werden, dass bei den Bedingungen, die für die Wahl der Wege zuständig sind, Headway-Aktivitäten ausgeschlossen werden.

Anhang

A Tabellen

Instanz	OD-Paar		$(ILP4)/(ILP5)$			$(ILP6)/(ILP8)$	
	u	v	M_e^1	M_e^2		M_v^1	M_v^2
				M_u	M_v		
1034	43	6	2955510	35730	31524	2955510	27066
	30	13	2955510	36174	29034	2955510	28711
	18	49	2955510	25590	24780	2955510	15576
	49	36	2955510	24762	30265	2955510	26826
	37	42	2955510	30522	36006	2955510	26388
2828	8	30	2955510	31278	36174	2955510	30024
	44	9	2955510	35424	32988	2955510	23130
	11	27	2955510	30846	28854	2955510	28854
	11	30	2955510	30846	36174	2955510	29520
	17	33	2955510	30900	30636	2955510	28045
	20	42	2955510	28062	36006	2955510	26262
	29	49	2955510	32838	24780	2955510	18588
4222	4	19	2955510	30522	27372	2955510	26838
	51	12	2955510	25062	30486	2955510	24252
	14	32	2955510	29028	30895	2955510	28386
	23	15	2955510	27510	24636	2955510	22218
	21	46	2955510	28062	34422	2955510	22380
	49	23	2955510	24762	27510	2955510	23460
	26	49	2955510	29070	24780	2955510	15006
	38	39	2955510	30978	30888	2955510	29340
7231	40	3	2955510	36480	30919	2955510	29472
	6	31	2955510	31525	31374	2955510	30036
	35	7	2955510	30588	32220	2955510	26328
	30	8	2955510	36174	31296	2955510	25878
	25	9	2955510	30078	32988	2955510	25758
	11	35	2955510	30846	30606	2955510	28698
	43	19	2955510	35730	27372	2955510	23064
	30	25	2955510	36174	25524	2955510	25260
	43	33	2955510	35730	30636	2955510	25440
	37	42	2955510	30522	36006	2955510	26388

Instanz	OD-Paar		$(ILP4)/(ILP5)$			$(ILP6)/(ILP8)$	
	u	v	M_e^1	M_e^2		M_v^1	M_v^2
				M_u	M_v		
8939	38	2	2955510	30978	31350	2955510	29586
	3	25	2955510	30936	25524	2955510	26496
	4	21	2955510	30522	28062	2955510	27162
	24	5	2955510	25926	30366	2955510	26262
	7	28	2955510	32100	29784	2955510	29634
	33	8	2955510	30636	31296	2955510	25944
	47	14	2955510	34140	28986	2955510	26700
	17	44	2955510	30900	35442	2955510	26316
	44	19	2955510	35424	27372	2955510	23250
	23	33	2955510	27510	30637	2955510	25350
	24	29	2955510	25926	32838	2955510	25692
	27	25	2955510	28854	25524	2955510	25428
	28	46	2955510	29784	34422	2955510	25764
	51	30	2955510	25062	36174	2955510	28374
	43	41	2955510	35730	36264	2955510	26808
7859	10	2	2955510	32988	31351	2955510	29850
	26	8	2955510	29070	31296	2955510	25728
	41	8	2955510	36246	31296	2955510	25998
	50	8	2955510	24924	31296	2955510	24540
	20	10	2955510	28062	32988	2955510	22026
	28	10	2955510	29784	32988	2955510	25554
	27	18	2955510	28854	25590	2955510	25806
	29	18	2955510	32838	25590	2955510	28998
	19	31	2955510	27390	31374	2955510	29580
	21	23	2955510	28062	27510	2955510	21918
	49	21	2955510	24762	28062	2955510	24372
	23	37	2955510	27510	30498	2955510	26166
	48	24	2955510	24751	26856	2955510	23568
	32	49	2955510	30876	24780	2955510	17544
	43	33	2955510	35730	30636	2955510	25440

Instanz	OD-Paar		$(ILP4)/(ILP5)$			$(ILP6)/(ILP8)$	
	u	v	M_e^1	M_e^2		M_v^1	M_v^2
				M_u	M_v		
7196	13	2	2955510	29052	31351	2955510	29382
	3	14	2955510	30936	28986	2955510	29742
	17	3	2955510	30900	30918	2955510	29100
	5	18	2955510	30384	25590	2955510	26442
	50	8	2955510	24924	31296	2955510	24540
	36	10	2955510	30540	32988	2955510	25962
	48	12	2955510	24751	30486	2955510	23208
	35	13	2955510	30588	29034	2955510	29016
	36	13	2955510	30540	29034	2955510	29154
	17	41	2955510	30900	36264	2955510	29412
	48	17	2955510	24751	28974	2955510	27144
	25	21	2955510	30078	28062	2955510	26202
	30	21	2955510	36174	28062	2955510	26178
	34	28	2955510	30486	29784	2955510	28968
	44	35	2955510	35424	30606	2955510	25890
	42	40	2955510	35988	35934	2955510	29730
	42	41	2955510	35988	36264	2955510	29874

Tabelle A.1: Die M's für $(ILP4)/(ILP5)$ und $(ILP6)/(ILP8)$

Ergebnisse	Rechner 1	Rechner 2	Gurobi 3.0.0	Gurobi 4.5.1
Tabelle 5.5		✓		✓
Tabelle 5.8		✓		✓
Tabelle 5.9		✓		✓
Tabelle 5.10		✓		✓
Tabelle 5.9		✓		✓
Tabelle 5.12		✓		✓
Tabelle A.3	✓		✓	
Tabelle A.4		✓		✓

Tabelle A.2: Übersicht über der Nutzung der beiden Rechner und der beiden Gurobiversionen
Rechner 1 = 64-Bit-Rechner mit 12GB Arbeitsspeicher und 3 Prozessoren vom Typ "Dual Core AMD Opteron(tm) Processor 275" mit einer Taktfrequenz von jeweils 2200 MHz
Rechner 2 = 64-Bit-Rechner mit 9.8GB Arbeitsspeicher und 1 Prozessor vom Typ "SunFire X4200, AMD Opteron 2.2 GHz"

Notation A.1:
In den Spalten von Tabelle A.3 stehen:

- ① die Kürzeste-Wege-Methode und die Startlängen K_a
- ② der Zielfunktionswert z, den die Heuristik beim Abbruch liefert
- ③ die Differenz des Wertes aus ② und der unteren Schranke LB_2 in %
- ④ die Gesamtzeit, die die Heuristik gebraucht hat
- ⑤ die Anzahl der Iterationen, die die Heuristik gebraucht hat (maximal 99 Iterationen)
- ⑥ die Zeit pro OD-Paar, berechnet durch $\frac{④}{⑤ \cdot \#\text{ OD-Paare}}$
- ⑦ die Differenz des Zielfunktionswertes aus Iteration 1 zu der unteren Schranke LB_2 in %
- ⑧ die Differenz des Zielfunktionswertes aus Iteration 2 zu der unteren Schranke LB_2 in %
- ⑨ Zielfunktionswert in Iteration 50 beziehungsweise bei #Iterationen < 50 der Wert z aus ②

Instanz		①	②	③	④	⑤	⑥	⑦	⑧	⑨
BG2H7_1	LA FH	4.429344E9	0.4991	238 m 13 s	99	0.0223 s	2.2771	0.6687	4.42937724E9
	UA FH	4.42938312E9	0.5	249 m 29 s	99	0.0233 s	1.9477	0.6559	4.42941462E9
	halb FH	4.42945908E9	0.5017	250 m 10 s	99	0.0234 s	2.0046	0.5666	4.42946262E9
	LA TMQ	4.43330664E9	0.589	5 m 49 s	7	0.0077 s	1.658	0.6338	z
	UA TMQ	4.4364048E9	0.6593	7 m 41 s	11	0.0065 s	1.7089	0.7047	z
	halb TMQ	4.43640804E9	0.6593	14 m 31 s	25	0.0054 s	1.7071	0.7184	z
BGX_1	LA FH	2.098449714E10	2.91	1369 m 44 s	99	0.0383 s	9.3061	3.1076	2.098467114E10
	UA FH	2.098364982E10	2.9058	1552 m 12 s	99	0.0434 s	8.6248	3.0985	2.098384884E10
	halb FH	2.09801511E10	2.8887	1370 m 33 s	99	0.0383 s	8.3872	3.1052	2.098063806E10
	LA TMQ	2.099425938E10	2.9579	240 m 31 s	99	0.0067 s	10.148	3.2229	2.099425938E10
	UA TMQ	2.09914155E10	2.9439	30 m 29 s	11	0.0077 s	9.2418	3.1823	z
	halb TMQ	2.099169414E10	2.9453	257 m 44 s	99	0.0072 s	9.0792	3.1925	2.099169414E10
BGX_2	LA FH	4.68216096E9	0.5701	127 m 45 s	99	0.0242 s	2.0063	0.6738	4.6821624E9
	UA FH	4.68256458E9	0.5788	121 m 57 s	99	0.0231 s	1.7859	0.7164	4.68278538E9
	halb FH	4.68182442E9	0.5629	129 m 28 s	99	0.0245 s	1.5912	0.6952	4.68183048E9
	LA TMQ	4.68348438E9	0.5985	5 m 19 s	11	0.0091 s	2.1761	0.6518	z
	UA TMQ	4.6833273E9	0.5951	30 m 55 s	99	0.0059 s	1.9987	0.6609	4.6833273E9
	halb TMQ	4.68445212E9	0.6193	29 m 43 s	99	0.0056 s	1.9402	0.6611	4.68445212E9
BGX_3	LA FH	155280.0	0	24 s	2	0.4138 s	0	0	z
	UA FH	155280.0	0	26 s	2	0.4830 s	0	0	z
	halb FH	155280.0	0	34 s	2	0.5862 s	0	0	z
	LA TMQ	155280.0	0	22 s	2	0.3793 s	0	0	z
	UA TMQ	155280.0	0	25 s	2	0.4310 s	0	0	z
	halb TMQ	155280.0	0	22 s	2	0.3793 s	0	0	z
BG3X_1	LA FH	359220.0	0	21 s	2	0.0493 s	0	0	z
	UA FH	359220.0	0	23 s	2	0.0540 s	0	0	z
	halb FH	359220.0	0	21 s	2	0.0493 s	0	0	z
	LA TMQ	359220.0	0	21 s	2	0.0493 s	0	0	z
	UA TMQ	359220.0	0	21 s	2	0.0493 s	0	0	z
	halb TMQ	359220.0	0	23 s	2	0.0540 s	0	0	z
BK2X_1	LA FH	1.721876862E10	2.2415	839 m 31 s	99	0.0322 s	8.5077	2.4644	1.721918892E10
	UA FH	1.721578188E10	2.2238	962 m 46 s	99	0.0369 s	7.8054	2.4532	1.72158363E10
	halb FH	1.72218498E10	2.2598	1016 m 31 s	99	0.0390 s	7.8466	2.4638	1.722296376E10
	LA TMQ	1.72313172E10	2.316	24 m 17 s	16	0.0058 s	9.4062	2.6123	z
	UA TMQ	1.72390986E10	2.3622	19 m 33 s	13	0.0057 s	8.6659	2.6117	z
	halb TMQ	1.723530006E10	2.3397	15 m 44 s	10	0.0060 s	8.6254	2.5638	z
BK2X_2	LA FH	4.46736618E9	0.6993	85 m 51 s	99	0.0188 s	2.5286	0.778	4.4678538E9
	UA FH	4.4689839E9	0.7357	82 m 43 s	99	0.0181 s	2.3971	0.8274	4.46925078E9
	halb FH	4.4678913E9	0.7111	69 m 16 s	99	0.0152 s	2.5141	0.8935	4.4679351E9
	LA TMQ	4.4703267E9	0.766	1 m 55 s	7	0.0059 s	2.6735	0.8186	z
	UA TMQ	4.47090324E9	0.779	2 m 10 s	10	0.0047 s	2.5838	0.8405	z
	halb TMQ	4.47090324E9	0.779	2 m 15 s	10	0.0049 s	2.5766	0.8405	z
BK2H7_1	LA FH	1.712259768E10	1.9703	975 m 18 s	99	0.0377 s	7.494	2.1843	1.712289156E10
	UA FH	1.711741872E10	1.9394	991 m 4 s	99	0.0383 s	7.6083	2.2104	1.711757334E10
	halb FH	1.71209124E10	1.9602	929 m 56 s	99	0.0359 s	6.9856	2.2317	1.71234243E10
	LA TMQ	1.71331011E10	2.0328	24 m 8 s	14	0.0066 s	8.0704	2.2673	z
	UA TMQ	1.713366762E10	2.0362	16 m 56 s	9	0.0072 s	7.6947	2.2453	z
	halb TMQ	1.712898714E10	2.0083	19 m 21 s	11	0.0067 s	7.4639	2.288	z
BK2H7_2	LA FH	4.42485948E9	0.4754	97 m 37 s	99	0.0218 s	1.8887	0.547	4.42486458E9
	UA FH	4.42468074E9	0.4713	82 m 38 s	99	0.0185 s	1.8386	0.5802	4.42468074E9
	halb FH	4.42470264E9	0.4718	97 m 34 s	99	0.0218 s	1.7448	0.5298	4.42470264E9
	LA TMQ	4.42723278E9	0.5292	2 m 36 s	8	0.0072 s	1.8033	0.5945	z
	UA TMQ	4.4277867E9	0.5418	2 m 38 s	8	0.0073 s	1.9338	0.5668	z
	halb TMQ	4.4269755E9	0.5234	2 m 37 s	8	0.0072 s	1.9335	0.592	z
SH7_1	LA FH	927600.0	0	26 s	99	0.0060 s	24.7089	2.0699	927600.0
	UA FH	927600.0	0	23 s	99	0.0053 s	15.1358	0	927600.0
	halb FH	927600.0	0	14 s	99	0.0032 s	15.7827	0	927600.0
	LA TMQ	927600.0	0	0 s	3	0.0000 s	10.3493	0	z
	UA TMQ	927600.0	0	1 s	3	0.0076 s	4.1397	0	z
	halb TMQ	927600.0	0	1 s	3	0.0076 s	4.1397	0	z
AMX_1	LA FH	4.7690436E7	0.3803	78 m 36 s	99	0.0196 s	9.8494	0.3998	4.7690436E7
	UA FH	4.7690436E7	0.3803	75 m 32 s	99	0.0189 s	10.0849	0.413	4.7690436E7
	halb FH	4.7691264E7	0.382	75 m 5 s	99	0.0188 s	10.187	0.4101	4.7691816E7
	LA TMQ	4.7812212E7	0.6366	3 m 19 s	10	0.0082 s	4.1259	0.9642	z
	UA TMQ	4.7771832E7	0.5516	2 m	6	0.0087 s	4.0481	1.378	z
	halb TMQ	4.7771832E7	0.5516	2 m	6	0.0087 s	4.0481	1.378	z
AMX_2	LA FH	4.7690436E7	0.3803	48 m 36 s	99	0.0121 s	10.0447	0.4101	4.7690436E7
	UA FH	4.7691324E7	0.3822	52 m 31 s	99	0.0131 s	9.4467	0.4006	4.7691324E7
	halb FH	4.769214E7	0.3839	49 m 23 s	99	0.0123 s	8.8669	0.3969	4.769214E7
	LA TMQ	4.7714322E7	0.4306	1 m 38 s	8	0.0050 s	4.1328	1.3029	z
	UA TMQ	4.7764614E7	0.5364	1 m 52 s	9	0.0051 s	4.055	1.4145	z
	halb TMQ	4.7764614E7	0.5364	1 m 50 s	9	0.0050 s	4.055	1.4145	z
AMX_3	LA FH	4.7692152E7	0.3839	24 m 5 s	99	0.0060 s	9.9107	0.5117	4.7692152E7
	UA FH	4.769544E7	0.3908	24 m 3 s	99	0.0060 s	9.5674	0.4387	4.769544E7
	halb FH	4.7694792E7	0.3895	23 m 5 s	99	0.0060 s	10.6038	0.4791	4.7694792E7
	LA TMQ	4.7864088E7	0.7458	8 m 56 s	99	0.0020 s	4.1328	1.4619	z
	UA TMQ	4.7882472E7	0.7845	49 s	9	0.0022 s	4.055	1.5374	z
	halb TMQ	4.7882472E7	0.7845	49 s	9	0.0022 s	4.055	1.5374	z

Tabelle A.3: Ergebnisse der Heuristik unter Verwendung von Gurobi 3.0.0

Instanz	#OD	$(ILP4)$				$(ILP6)$			
		M_e^1		M_3^2		M_v^1		M_v^2	
		z	t	z	t	z	t	z	t
1034	5	56898_H	195 7200 7599 7601	56898_H	285 7200 7618 7619	56898_*	141 3845 3850 3852	56898_H	48 4155 4162 4162
2828	7	94320_*	862 7204 8101 8102	95328_H	38 7237 8187 8188	94320_H	32 7200 7207 7209	94320_H	29 7200 7207 7209
4222	8	169428_H	478 7195 8404 8406	169428_H	5442 7200 8329 8331	169428_H	106 7200 7206 7207	169428_H	62 7200 7205 7207
7231	10	191796_H	782 7200 9053 9055	192780_H	262 7200 9099 9101	191340_H	245 7200 7210 7212	191340_H	48 7200 7207 7210

Tabelle A.4: Zielfunktionswerte von $(ILP4)$ und $(ILP6)$ (also ohne untere Schranke LB)
_H - Wert durch Heuristik von Gurobi berechnet
_* - Wert durch Branching von Gurobi berechnet
Das Basis-Ereignis-Aktivitätsnetzwerk war AMX_3 mit 3614 Knoten und 4094 Kanten. Mit der OD-Datei OD_neu_386.txt als Basis wurden unabhängig voneinander verschiedene Teilmengen der OD-Paare ausgewählt und getestet.

NB Instanz	1034	2828	4222	7231	8939	7859	7196
# NB	26651	34218	37632	45203	63601	63270	70981
Flow-out NB	5	7	8	10	15	15	17
Flow-in NB	5	7	8	10	15	15	17
Flußerhaltung	18060	25284	28897	36120	54181	54180	61404
Timetabling NB	8224	8224	8224	8224	8224	8224	8224
t_u $\leq$-NB	219	304	248	530	524	480	656
t_v $\geq$-NB	137	391	246	308	641	355	662
LB_2-NB	1	1	1	1	1	1	1
# Variablen	24450	32981	36876	45412	66219	65889	74564
# Π-Variablen	3614	3614	3614	3614	3614	3614	3614
# q-Variablen	20826	29353	33246	41778	62575	62245	70916
# t_u-Variablen	5	7	8	10	15	15	17
# t_v-Variablen	5	7	8	10	15	15	17

Tabelle A.5: Anzahl der Nebenbedingungen von $(ILP5)$ aus Tabelle 5.9

NB Instanz	1034	2828	4222	7231	8939	7859	7196
# NB	8765	9136	8851	9534	10070	9358	10007
# Wege	5	7	8	10	15	15	17
Timetabling NB	8224	8224	8224	8224	8224	8224	8224
tk $\leq$-NB	535	904	618	1299	1830	1118	1765
LB_2-NB	1	1	1	1	1	1	1
# Variablen	4154	4525	4240	4923	5459	4747	5396
# Π-Variablen	3614	3614	3614	3614	3614	3614	3614
# z_k-Variablen	535	904	618	1299	1830	1118	1765
# t_k-Variablen	5	7	8	10	15	15	17
$\varnothing$ # Wege/OD-Paar	107	129,14	77,25	129,9	122	74,53	103,82

Tabelle A.6: Anzahl der Nebenbedingungen von $(ILP8)$ aus Tabelle 5.9

NB Instanz	1034	2828	4222	7231	8939	7859	7196
# NB	26650	34217	37631	45202	63600	63269	70980
Flow-out NB	5	7	8	10	15	15	17
Flow-in NB	5	7	8	10	15	15	17
Flußerhaltung	18060	25284	28897	36120	54181	54180	61404
Timetabling NB	8224	8224	8224	8224	8224	8224	8224
$t_u \leq$-NB	219	304	248	530	524	480	656
$t_v \geq$-NB	137	391	246	308	641	355	662
LB_2-NB	0	0	0	0	0	0	0
# Variablen	24450	32981	36876	45412	66219	65889	74564
# Π-Variablen	3614	3614	3614	3614	3614	3614	3614
# q-Variablen	20826	29353	33246	41778	62575	62245	70916
# t_u-Variablen	5	7	8	10	15	15	17
# t_v-Variablen	5	7	8	10	15	15	17

Tabelle A.7: Anzahl der Nebenbedingungen von $(ILP4)$ aus Tabelle A.4

NB Instanz	1034	2828	4222	7231	8939	7859	7196
# NB	8764	9135	8850	9533	10069	9357	10006
# Wege	5	7	8	10	15	15	17
Timetabling NB	8224	8224	8224	8224	8224	8224	8224
$tk \leq$-NB	535	904	618	1299	1830	1118	1765
LB_2-NB	0	0	0	0	0	0	0
# Variablen	4154	4525	4240	4923	5459	4747	5396
# Π-Variablen	3614	3614	3614	3614	3614	3614	3614
# z_k-Variablen	535	904	618	1299	1830	1118	1765
# t_k-Variablen	5	7	8	10	15	15	17

Tabelle A.8: Anzahl der Nebenbedingungen von $(ILP6)$ aus Tabelle A.4

NB Instanz	BGX_3	BG3X_1
# Zsghskomp.	502	398
# Isolierte Knoten	43	23
# NB	52038	313664
Flow-out NB	29	213
Flow-in NB	29	213
Flußerhaltung	48227	308424
Timetabling NB	3596	3132
$t_u \leq$-NB	83	930
$t_v \geq$-NB	73	751
LB_2-NB	1	1
# Variablen	39533	252575
# Π-Variablen	1706	1471
# q-Variablen	37769	250678
# t_u-Variablen	29	213
# t_v-Variablen	29	213

Tabelle A.9: Anzahl der Nebenbedingungen von $(ILP6)$ von BG3X_1 und BGX_3

NB Instanz	BGX_3	BG3X_1
# NB	3665	3638
# Wege	29	213
Timetabling NB	3596	3132
$tk \leq$-NB	39	292
LB_2-NB	1	1
# Variablen	1774	1976
# Π-Variablen	1706	1471
# z_k-Variablen	39	292
# t_k-Variablen	29	213
$\varnothing$ # Wege/OD-Paar	1,34	1,37

Tabelle A.10: Anzahl der Nebenbedingungen von $(ILP8)$ von BG3X_1 und BGX_3

B Der Code

Der vollständige Code der Heuristik und der anderen selbstgeschriebenen Programme, sowie die Ergebnisse der Heuristik und der ganzzahligen linearen Programme sind auf der beiliegenden DVD zu finden.

Abbildungsverzeichnis

Literaturverzeichnis

[Bol90] BOLLOBÁS, Béla: *Graph Theory - An Introductory Course*. Third. Springer Verlag, 1990

[Bre92] BRETZ, Manfred: *Algorithmen und Berechenbarkeit*. Friedr. Vieweg Verlag & Sohn Verlag, 1992

[CLRS07] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald ; STEIN, Clifford: *Algorithmen - Eine Einführung*. second. Oldenbourg Wissenschaftsverlag GmbH, 2007

[Fou09] FOURNIER, Jean-Claude: *Graph Theory and Applications*. Iste Ltd, John Wiley & Sons, Inc., 2009

[FP93] FANG, Shu-Cherng ; PUTHENPURA, Sarat: *Linear Optimization and Extensions - Theory and Algorithms*. Pearson US Imports & PHIPEs, 1993

[GNU11] GNU: *GNU General Public License*. `http://www.gnu.org/licenses/gpl-3.0.html`. Version: 2011

[Gur11] GUROBI: *Gurobi Optimization*. `http://www.gurobi.com/`. Version: 2011

[HK06] HAMACHER, Horst W. ; KLAMROTH, Kathrin: *Lineare Optimierung und Netzwerkoptimierung*. second. Friedr. Vieweg Verlag & Sohn Verlag, 2006

[Jav11] JAVA: *Java*. `http://www.oracle.com/technetwork/java/index.html`. Version: 2011

[KNW05] KRUMKE, Sven O. ; NOLTEMEIER, Hartmut ; WIRTH, Hans-Christoph: *Graphentheoretische Konzepte und Algorithmen*. Teubner, 2005

[Lin11] LINTIM: *LinTim*. `http://lintim.math.uni-goettingen.de/`. Version: 2011. – Projekt der Georg-August-Universität Göttingen

[May05] MAY, Wolfgang: *Einführung in Datenbanken - Wintersemester 2004/2005*. `http://www.dbis.informatik.uni-goettingen.de/Teaching/DB-WS0405/`. Version: 2005. – Vorlesungsskript der Georg-August-Universität Göttingen

[NC11] NAVEH, Barak ; CONTRIBUTORS: *JGraphT*. `http://www.jgrapht.org/`. Version: 2011

[Sch06] SCHÖBEL, Anita: *Optimization in Public Transportation*. Springer, 2006

[Sch07] SCHÖBEL, Anita: *Einführung in die Optimierung - Sommersemester 2005*. 2007. – Vorlesungsskript der Georg-August-Universität Göttingen

[Sch09] SCHACHTEBECK, Michael: *Delay Management in Public Transportation: Capacities, Robustness, and Integration*, Georg-August-Universität Göttingen, Dissertation, 2009

[Sch10] SCHÖBEL, Anita: *Optimization Models in Public Transportation*. 2010. – Skript der Georg-August-Universität Göttingen

[Sch11] SCHMIDT, Marie: *Integrating Routing Decisions in Network Problems*, Georg-August-Universität Göttingen, Dissertation, 2011

[Sie11] SIEBERT, Michael: *Integrating of Routing and Timetabling in Public Transportation*, Georg-August-Universität Göttingen, Diplomarbeit, 2011

[SS10] SCHMIDT, Marie ; SCHÖBEL, Anita: The Complexity of Integrating Routing Decisions in Public Transportation Models. In: ERLEBACH, Thomas (Hrsg.) ; LÜBBECKE, Marco (Hrsg.): *ATMOS 2010 - 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, 2010 (Dagstuhl Seminar Proceedings). – ISBN 978–3–939897–20–0. – http://drops.dagstuhl.de/opus/volltexte/2010/2757/